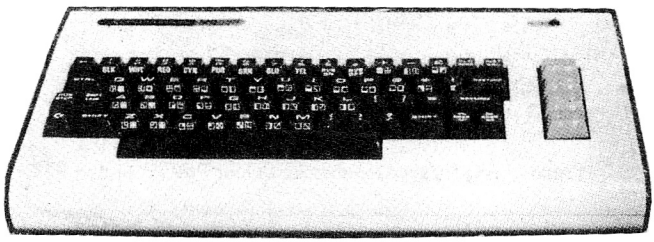
GETTING ACQUAINTED WITH YOUR VIC 20

TIM HARTNELL

GETTING ACQUAINTED WITH YOUR VIC 20



BY
TIM HARTNELL

Published in Great Britain by:

INTERFACE,

44 – 46 Earls Court Road,
London W8 6EJ.

ISBN 0 907563 05 8

This book is based on material from ZX81 books by Tim Hartnell, converted for the VIC 20, with additional material, by Toni Baker.

All programs checked and entered by Graham Charlton & Mark Ramshaw

First printing - September 1981
Second printing - November 1981
Second edition - February 1982
Reprinting, with corrections and amendments - June 1982
Reprinted - September 1983

Printed in England by Commercial Colour Press, London E7 OHX.

Foreword

The VIC 20 is a special computer, and you need a special book to help you get the most out of your computer. This, I believe, is such a book.

It will lead you, step by simple step, from the absolute basics of programming your VIC, introducing you to many of the functions and commands available in the BASIC language, through the splendid sound, music and colour possibilities of the VIC, to the point where you'll soon be writing and adapting programs by yourself.

While the book may just seem to be a collection of games for the Commodore VIC — and there are more than 60 games given in the book — closer inspection will show you that following through the explanation given for each game, and computer function, will add to your knowledge of the VIC, of BASIC, and of computers in general.

However, the book has much more than games. If you're wondering what to do with your new computer now that you have it — apart from playing games — you'll find a host of ideas including several which may help you make a bit of money with your VIC. And if you're a parent or teacher, you'll find the section THE VIC AS TEACHER an aid in making the most effective use of the computer in the classroom. Tim Hartnell has outlined some quite useful routines for educational applications.

The book is a worthwhile resource to make sure you make the most of your computer. And you'll never feel quite the same about your VIC after surviving a round of FRENZY, or listening to your computer composing a symphony.

ZOË SHEPHERD,
London, September, 1981

OTHER PUBLICATIONS:

GETTING ACQUAINTED WITH YOUR ACORN ATOM
GETTING ACQUAINTED WITH YOUR ZX81
MASTERING MACHINE CODE ON YOUR ZX81
49 EXPLOSIVE GAMES FOR THE ZX81
34 AMAZING GAMES FOR THE 1K ZX81
GATEWAY GUIDE TO THE ZX81 & ZX80

Abbreviations

Many of the keys on the VIC keyboard have symbols which cannot be printed. For this reason, and also for reasons of clarity, the following conventions will be used throughout the book.

Spaces in print statements will generally be written out in full and in small letters, eg PRINT "HELLO space" means there is one and only one space after the O in the word HELLO. Similarly PRINT "two spaces BYE" means that there are exactly two spaces between the first quote and the word BYE. In many cases this convention will be dropped if it is obvious how many spaces are needed, eg PRINT "BYE BYE THEN" would be preferable to PRINT "BYE space BYE space THEN".

Graphics symbols are also available from the VIC keyboard. The right hand graphics are obtained by using the shift key, and these will be written out as, for instance, shift A or shift Q or shift F. The left hand symbols are obtained by using the commodore key. These will be written as, for example, commodore Q or commodore K.

Many of the keys contain control characters which may be used in PRINT statements. We shall write the word which is printed on the key itself (ie not necessarily what appears on the screen) in CAPITAL LETTERS, thus, in a slightly lighter print. For instance PRINT "CLR HELLO" means the symbol obtained by pressing shift and the CLR/HOME key, followed by the word HELLO. There is no space between CLR and the letter H — if one is needed it will be written out in full. Other examples could be PRINT "RED H YEL E GRN L PUR L CYN O BLU", or PRINT "RVS ON DANGER RVS OFF."

Introduction

Welcome to this book.
If you work through it with your VIC turned on at the appropriate moments, you should learn enough about programming to write your own games programs. If you follow the advice given on "building a library" you'll also have a healthy set of neatly labelled cassettes with a choice of programs to please even your most boring friends.

The book assumes you would like to collect a number of different programs that work; that you want to know how and why they work; and — eventually — that you want to be able to create and develop your own programs. The book explains how the programs were written, developed, and made more complex and/or fun to run, from a simpler "core" program. By following through the programming, feeding the games and other programs into your computer and running them, you should learn a fair amount of BASIC without even trying.

Before you read the book, I suggest you glance through it to get an overall picture of the ground it covers. The more BASIC you know, the further you can read into the book without plugging in your VIC and running the programs.

The book was written with the idea that you would input each program when you came to it. This is why, for example, all the HUNT THE HURKLE-type games are not together. You're most unlikely to want to input six different versions of the same program in a row. The order of programs is also dictated by the need to put the simpler ones in first.

Tim Hartnell.

Random Numbers

Turn on your TV, and get the cursor on the screen. Our first program uses one of the most useful programming aids for games — the random number generator.

Actually, it is not a true random number generator, but it is close enough for our purposes. To generate a number between one and, say 10, you just input J = INT (10 * RND (1)) + 1. If you follow up this line with the command PRINT J, your computer will display the number (between one and 10) which it has generated. Notice that when you write a program, each line must be numbered. The computer executes each line from the lowest number to the highest.

Try this program first:

```
10 PRINT"NUMBER GENERATOR"
20 J=INT(10*RND(1))+1
30 PRINTJ;" ";
40 GOTO20
```

When you run this, you'll find the screen fills up with numbers, between 1 and 10, press the STOP key to stop the program.

There are many things you can learn from this program:

10 PRINT "NUMBER GENERATOR"

This line starts with a line number (as I mentioned before). You can use any number you like (between 1 and 9999), but you'll find working in multiples of 10 or 20 will give you enough room to add new lines in between others if you need to later. Line numbers sort themselves automatically into the correct order. When you ran the program, the VIC acted on the lowest numbered line (in this case 10) and obeyed the instruction PRINT and printed what was between the quote marks.

Pretty simple, basic stuff. So from this line you have learned the use of line numbers, and of the command PRINT.

The next line:

*20 J = INT (10 * RND (1)) + 1*

Lets look at how this line works very closely. RND (1) produces a random decimal number somewhere between zero and 0.999999.... The (1) in brackets does not determine the range of random numbers produced, instead it determines how the random number is produced. We shall look at this in a minute.

10 * RND (1) is therefore a decimal number between zero and 9.99999.... and this is where the function INT comes in.

INT of a number produces (for positive numbers) the part of the number to the left of the decimal point, so INT (3.5) is 3, and INT (0.6) is 0. (For negative numbers INT will give the part of the number to the left of the decimal point less one, so that INT (−0.6) is −1, and INT (−3.5) is −4).

Now, if 10 * RND (1) is a decimal number between 0 and 9.99999.... then INT (10 * RND (1)) is an integer (or whole number) between 0 and 9. Adding one produces our random number between one and ten.

Now look at the first part of the line. This is 20 J =....

This tells the computer to assign the value generated to the integer variable J. Then when, in the next line, you ask the computer to PRINT J, it prints the number which has been assigned to J. Don't worry if you can't understand this, it will become clear in due course.

The next line:

30 PRINT J; " ";

This line tells the computer to print the number which it has assigned to J. The semi-colon after the J, and after the second set of quote marks, ensures that the computer will print the values for J one after another, instead of starting a new line for each one.

Try typing 30 PRINT, J, Using ENTER, put this into the program in place of the original line 30. Run the program, and you'll notice the numbers printed in neat little columns. The comma divides the screen up into two parts, and when it reads a comma, automatically goes to the start of the next half of the screen. The semi-colon tells the VIC to go on and print the next J, leaving a space, without starting a new line.

The fourth line of our original program:

40 GOTO 20

The computer runs through the program in order, carries out the instructions in each line, and then stops. In this case, when it gets to the end of the program, line 40 tells it to go back to line 20. It does so, then runs through the program again, and again finds the instructions to go back to line 20. It stops only when you run out of screen space.

Now let's look at the (1) in brackets in line 20. Let's see what this does. Change line 20 to J = INT(10*RND(2)) + 1 and RUN it again. As you can see this makes no difference at all. Change it to J = INT (10*RND

(10Ø)) + 1 and then RUN it — again there is no difference. (Remember to press the STOP key to end the program). In fact as long as the number in brackets is POSITIVE (this also means not zero by the way) then RND will always work in the same way.

Now change it to J = INT(1Ø*RND(Ø)) + 1 and RUN it again. Rather stunning eh? Each "random" number is the same, and in fact is the same as the last random number produced. RUN it again, and again, and again.

Add the line 5 J = RND (1). Now run it over and over again. Each time you RUN it all of the "random" numbers will be the same, but the value of this number is different every RUN.

Here's why.

RND (1)	(or RND (any positive number)) is different every time.
RND (Ø)	generates the same number which was generated last time RND was used.

Let's look at what happens if the number in the brackets is negative. Leaving line 5 in place, change line 2Ø to J = INT(1Ø*RND(−5)), and RUN the program over and over again. You see that the number is the same every time, and even with line 5 in place it is the same every RUN. Line 5 makes no difference, so delete it.

Change line 2Ø to J = INT(1Ø*RND(−5Ø)) and see what happens.

RND (−x)	first of all re-seeds the random number generator with x, and then generates a random number from it.
RND (+x)	first of all re-seeds the random number generator with a mathematical manipulation of the previous seed, and then generates a new random number from it. The value of x is irrelevant.
RND (Ø)	does not re-seed the random number generator at all, but generates a random number from the OLD seed.

A useful trick to know is that if you make the first line of a program J = RND(−TI) then RND (1) will from then on be completely random.

Now, you've learned that a letter, such as J, can be assigned a numerical value. What is known as a 'string' (a letter, or a letter followed by a number, or two letters, followed by a $ sign, like A$ or BC$) can be assigned to a word or words (or any combination of letters, symbols and numbers, but we'll stick to words for the moment).

Clear the RAM with the instruction NEW, and input the following program:

```
10 PRINT"NUMBER GENERATOR"
20 PRINT"WHAT IS YOUR NAME?"
30 INPUTA$
40 PRINT"OK,";A$;",I AM"
45 PRINT"GOING TO PRINT"
50 PRINT"SOME RANDOM NUMBERS   BETWEEN ONE"
60 PRINT"AND TEN.PRESS RETURN"
70 INPUTB$
80 J=INT(10*RND(1))+1
90 PRINTJ;
100 GOTO80
```

Try running this program. You'll find the line 20 asks your name, and then accepts it in line 30, assigning the string A$ to your name. The computer then uses your name (A$) in the next line, line 40.

The other thing to note about this program is the line 60 which tells you to press the RETURN key to continue. The string in line 70 (B$) is just a way to tell the computer to stop and wait for RETURN to be pressed. This feature is a very useful one in games, and we will be using it time and time again.

The clear screen facility is very useful here. PRINT "CLR" you will find, clears the screen of everything that preceded it on the program, but doesn't (as will become important later) 'forget' what has been assigned to the string, A$. To get the special CLR character in quotes, hold down SHIFT and press the CLR/HOME key. That is the computer still remembers your name (A$), so long as you don't wipe the program with NEW or turn off the power, or go back into the 'command mode' (when you can see the whole program listed on the screen, numbers and all).

We can let the computer count how many times it goes around the loop (that is, how many times it executes the lines 80,90 and 100). Add the following:

```
76 K%=0
85 K%=K%+1
95 IFK%>30THEN END
```

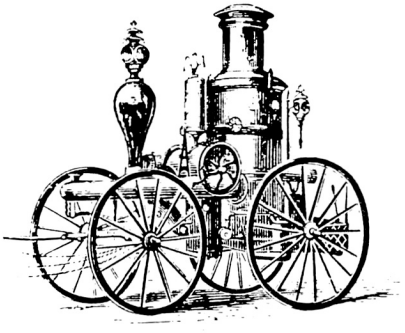
If the variable K% confused you don't worry. The % sign just means K is an INTEGER variable.

Integer variables can only store whole numbers between − 32768 and + 32767, but they are faster, and need less memory to store. Since J was an integer we would have called it J% if we'd have wished.

Now run it, you should find the program prints 31 numbers between 0 and 10 and then stops. It does this by assigning the value of zero to the number variable in line 76, then adds one to K% each time round (so the first time it is, in effect, LET space K% = 0 + 1, the second time LET space K % = 1 + 1, the third time LET space K % = 2 + 1, and so on) until K% is bigger than 30, when the computer ENDS. This counting and terminating feature is a very useful one.

IF/THEN; In line 95(IF K%> 30 THEN END) we used one of the computer's facilities to make a decision, and act on it. This IF/THEN is a very useful command in BASIC. The form of an IF/THEN line is line number, IF something (such as IF Z = 30, or IF A = B) followed by THEN and another command. In BASIC the other command can be anything (such as a line number to go to, LET S = 2, or END).

Let's get to our first real program.



Simple Decision Makers

Input the following:

```
10 PRINT,"DECISION       MAKER"
20 PRINT
30 PRINT
40 PRINT"THINK OF A QUESTION   AND"
50 PRINT"PRESS RETURN FOR A"
60 PRINT"DECISION ON IT"
70 INPUTQ$
80 J=INT(3*RND(1))
90 IFJ=0THENPRINT,"YES"
100 IFJ=1THENPRINT,"NO"
110 IFJ=2THENPRINT,"MAYBE"
120 END
```

Run this program a few times. You'll see how the comma in lines 10, 90, 100 and 110 sets the words away from the left hand side of the screen, and how lines 20 and 30 (with just the command PRINT, with nothing following it) put a space between the printout of lines 10 and 40.

Now comes the creative bit. To dress up this program, you can do at least three things: (1) Let the computer ask for, and use your name; (2) Use "CLR" to make the presentation cleaner, and (3) Allow you more than one go. Try and amend the program (by slipping things in between the numbered lines, and/or by changing some lines) until you can do these three things — before reading on. Cover up the following program until you've had a go at altering the original program.

This is just an example of how to modify the program. There are many, many other ways to achieve similar ends.

```
10 PRINT,"DECISION      MAKER"
20 PRINT
25 PRINT"WHAT IS YOUR NAME?"
30 INPUTA$
40 PRINT"□THINK OF A QUESTION,  ";A$;"."
50 PRINT"PRESS RETURN,AND I"
60 PRINT"WILL MAKE A DECISION  ON IT."
70 J=INT(3*RND(1))
80 INPUT□$
90 IFJ=0THENPRINT,"YES"
100 IFJ=1THENPRINT,"NO"
110 IFJ=2THENPRINT,"MAYBE"
120 PRINT
130 PRINT"ANOTHER GO,";A$;
140 INPUTB$
150 IFB$="YES"THEN40
170 PRINT"□OK,";A$;",BYE BYE"
180 END
```

Note the way the computer is asked to check if B$ = "YES" (in line 150). If it finds that B$ is equal to YES, control goes back to line 30. There is no need to put in a line IF B$ = "NO" THEN 160 because the computer moves on until it comes to another instruction.

There is another refinement we could add to this program. Try and work out how to make the amendment before you read the program listing. If you decided you wanted the computer to give you two "NO" and two "YES" to every one "MAYBE", how would you amend it? Try to work this out (changes will be needed between lines 80 and 120) before you read on.

If you need five alternatives (two NO, two YES, and one MAYBE), you will need line 80 to generate five random numbers. For two of these the computer must print NO, and so on. To save writting IF J = 0 THEN PRINT "YES", and IF J = 1 THEN PRINT "YES", and IF J = 3 THEN PRINT "NO" and so on, we can use the line IF J < 2 THEN PRINT "YES".

But what about the "NO" answer? If you write, for line 100, IF J < 5 THEN PRINT "NO", the computer will print a YES with a NO underneath if the number 3 or 4 is generated. You can overcome this as follows:

```
80 J=INT(5*RND(1))
90 IFJ<2THENPRINT"YES"
100 IFJ>2THENPRINT"NO"
110 IFJ=2THENPRINT"MAYBE"
```

A little later on we'll work out a more sophisticated DECISION MAKER program, but for now let's do something a little more interesting — play Russian roulette.

Russian Roulette

The principle of the game is simple. You have a pistol with six chambers, only one of which contains a bullet. You spin the chamber, pull the trigger, and ... either bang, or click. Already you are thinking "Aha, the random number generator, one in six." The only major decision to make is how many shots you are going to have before ending the game. Enter the following program, run it a few times, then come back to this book for a discussion on some of the features of it.

```
10 PRINT"RUSSIAN ROULETTE"
30 PRINT
40 J=0
50 INPUT"PRESS RETURN TO FIRE";T$
60 PRINT"□"
80 J=J+1
90 G=INT(6*RND(1))
100 IFG<5THENPRINT"CLICK"
110 IFG=5THEN140
120 IFJ=10THEN160
130 GOTO50
140 PRINT"BANG...";
150 GOTO140
160 PRINT"YOU HAVE SURVIVED"
170 END
```

Look at line *50 — INPUT "PRESS RETURN TO FIRE"; T$.* This first of all prints PRESS RETURN TO FIRE? on the screen, and then INPUTS T$ normally on the same line. This useful feature saves having to use both a PRINT and an INPUT statement.

Notice this program uses the GOTO (in lines 110,120,130 amd 150) command to either give you another 'shot', print BANG... to fill the screen, or to print YOU HAVE SURVIVED. Remember to pres the STOP key to end the game if BANG... is being printed. Line 40 sets J = 0 at the beginning, and line 80 adds one to it each time you run through until (unless you've shot yourself) you've been through ten times, that is when J = 10. Line 120 checks the value of J and either sends you back to 50 or advance to 160. Now, add the following line to the program and run it a few times:

```
95 PRINTG,J'
```

This added line displays the number that line 90 has generated and, to the right, the number of the run (ie J) that you are on. After you've run it a few times, erase line 95. Before you read on, try to amend the program (just by adding lines between the present ones) to allow the program to give you (a) the option not to play at all; and (b) the chance for subsequent games if you manage to survive.

```
30 INPUT"DO YOU WANT TO PLAY";A$
34 IFA$="NO"THENEND
```

and further down:

```
162 PRINT"DO YOU WANT ANOTHER GO"
165 INPUTB$
166 IFB$="YES"THEN40
```

You may well have modified the program differently, but as long as it works, it doesn't matter exactly how you do it.

Now, the following is a very important point for the development of games' programs. The best way I've found to work out games is to set up the 'core' of the game first (in the case of RUSSIAN ROULETTE, the core would be the first version of the game). Having set up this core, and made sure it works, I then proceed to elaborate the game to make it more fun to play.

You may have felt that the running of the game in its present form is a little unsatisfactory, and that when line 95 was included (PRINT G,J) it was a bit more interesting. So add the following line:

```
95 PRINT"THAT WAS SHOT";J
```

and run the program. That certainly makes it a bit more interesting. Before reading on, I want you to try and write a new version of line 95 which will tell how many shots you've left to reach 10, rather than just telling you the number of the current shot. One way of doing it:

```
95 PRINT10-J;"SHOTS TO GO"
```

Now, let us have a second look at line 34. Instead of just stopping when the player decides not to play, why not let the computer respond more definitely, say by printing the word CHICKEN to fill the screen. Try and work out how to do this. For a start, you'll have to change the END statement in line 34 into a GOTO statement.

One way of doing it:

```
34 IFA$="NO"THEN180
180 PRINT"CHICKEN...";
181 GOTO180
```

This version prints a most interesting pattern of the word CHICKEN. We'll use this pattern-effect to produce a new program shortly, but for now, add a few more lines to the program so that it uses the player's name. Once you've done that, and before you read my version below, try to write in a line so the computer won't print...SHOTS TO GO on the shot which fills the screen with BANG... This is my final version of the program:

```
10 PRINT,"RUSSIAN        ROULETTE"
20 PRINT
24 INPUT"WHAT IS YOUR NAME";A$
27 PRINT
28 PRINT
30 PRINT"DO YOU WANT TO PLAY,  ";A$
32 INPUTB$
33 PRINT"□"
34 IFB$="NO"THEN180
40 J=0
50 INPUT"PRESS RETURN TO FIRE";T$
70 PRINT"□"
80 J=J+1
90 G=INT(6*RND(1))
95 PRINTA$;",";10-J;"SHOTS TO GO"
96 PRINT
97 PRINT
100 IFG<5THENPRINT"CLICK"
```

```
105 PRINT
110 IFG=5THEN140
120 IFJ=10THEN160
130 GOTO50
140 PRINT"BANG...",
150 GOTO140
160 PRINT"YOU HAVE SURVIVED    ,";A$
162 PRINT"DO YOU WANT ANOTHER GO?"
164 INPUTC$
165 PRINT"□"
166 IFC$="YES"THEN40
180 PRINT"CHICKEN...";
190 GOTO180
```

Now, before we leave Russian roulette games, we'll look at one final version that introduces you to a new command, and shows a slightly different conversational approach:

```
10 PRINT"RUSSIAN ROULETTE (3)"
20 PRINT
25 PRINT"HI,GUN TOTER! YOU HAVE";
30 PRINT"A PISTOL WITH ONE     BULLET IN IT."
35 PRINT"YOU WILL SPIN THE     CHAMBER"
40 PRINT"AND FIRE 10 TIMES."
50 INPUT"ARE YOU GAME TO PLAY";A$
60 PRINT
65 IFA$="NO"THEN185
67 J=0
70 INPUT"PRESS RETURN TO FIRE";T$
80 PRINT"□"
85 PRINT
90 PRINT
95 PRINT
100 J=J+1
105 G=INT(6*RND(1))
110 PRINT"SHOT NUMBER";J
115 IFG<5THENGOSUB195
120 IFG=5THEN220
125 IFJ=10THEN135
130 GOTO70
```

```
135 PRINT"□"
140 PRINT
145 PRINT
150 PRINT"YOU SURVIVED."
155 PRINT"IF YOU WANT TO RISK   DEATH AGAIN"
160 PRINT"PRESS G FOR GO...OR S FOR STOP"
165 GETB$
170 IFB$="G"THEN67
180 IFB$<>"S"THEN165
185 PRINT"COWARD...";
190 GOTO185
195 H=INT(2*RND(1))
200 IFH=0THENPRINT"CLICK"
210 IFH=1THENPRINT"EMPTY CHAMBER"
215 RETURN
220 PRINT"BANG...";
225 GOTO220
```

Notice how GET was used instead of INPUT in line 165. GET can often be used instead of INPUT but only one character at a time may be entered, **and it is not necessary to press RETURN.**

Lines 170 and 180 will ensure that the GET line is run again if "G" or "S" is not pressed.

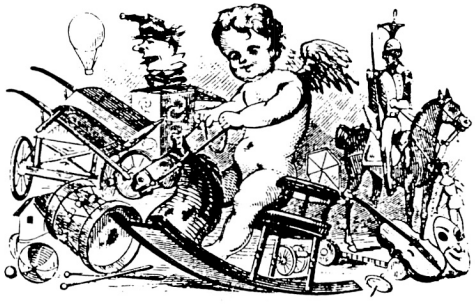
An interesting point about the GET statement is that it doesn't wait for a key to be pressed. The next line in the program continues immediately, and if no key is being pressed GET B$ will assign the empty string (" ") to B$.

You'll notice as you play this that an empty chamber (ie G < 5) gives you either CLICK or EMPTY CHAMBER, which, as you can see, is generated by the lines 195 to 215. The computer leaps to 195 from 115 on the command GOSUB 195.

Lines 195 through to 215 makes up a "subroutine". Whenever the computer comes across a GOSUB command, it goes to the line specified, and follows on until it comes to command RETURN. The computer then returns to the line after the GOSUB command. The GOSUB/RETURN is a very useful feature. We'll be using it again.

You will recall that when you first worked out how to get the computer to fill the screen with the word CHICKEN that it made quite an attractive pattern. We are going to look now at a slightly more developed form of pattern-making program, which uses the FOR/NEXT loop to limit the

number of times the sequence of letters or spaces is repeated. We will be looking at the FOR/NEXT loop in detail a little later.



Pattern Makers

The core of this program is simple. Feed it into your computer and after pressing RUN, input any six letters and/or spaces, then press RETURN.

```
10 INPUTA$
20 FORJ=1TO75
30 PRINTA$;
40 NEXTJ
```

You can run this program, which is surprisingly effective, using any combination of letters, numbers and symbols you like. You'll find that one or two spaces, instead of letters, enhance the pattern produced. Try a few more patterns, using spaces, letters like M and W, the $ sign, the numbers 6 and 9 and the graphics symbols.

Before reading on to see how I have done it, try to write around the core program you have to include the following features; (a) a title; (b) instructions; and (c) the chance to form another pattern without having to go back into the command mode and press RUN again. Cover up my version until you've tried your own.

One way (and there is an infinite number of ways you could have done it) is as follows:

```
1 PRINT"□       PATTERNS"
3 PRINT
6 PRINT"PRESS ANY COMBINATION OF 6 LETTERS ,SYMBOLS AND SPACES..."
```

```
8 PRINT
10 PRINT"...AND I WILL PRINT A PATTERN"
12 INPUTA$
15 PRINT"■"
20 FORJ=1TO75
30 PRINTA$;
40 NEXTJ
50 PRINT
60 PRINT"DO YOU WANT ANOTHER   GO?"
65 INPUTB$
80 IFB$="YES"THENRUN
90 PRINT"OK, SEE YOU LATER,    ARTIST"
100 END
```

This is quite an addictive program. The simplest combination of letters produces almost three-dimensional patterns. If you set this program up for one of your friends to try, be ready to wait a long, long time before you next have a go. For a colour pattern, add the lines

```
25 ONINT(8*RND(1))+1GOSUB120,140,160,180,200,220,240,260

120 PRINT"■";
130 RETURN
140 PRINT"■";
150 RETURN
160 PRINT"■";
170 RETURN
180 PRINT"■";
190 RETURN
200 PRINT"■";
210 RETURN
220 PRINT"■";
230 RETURN
240 PRINT"■";
250 RETURN
260 PRINT"■";
270 RETURN
```

Line 25 means if the random number expression is 1 then GOSUB 120, if the random number is 2 then GOSUB 140, and so on up to GOSUB 260.

Now look at the subroutines themselves. You will need to use the CTRL key to get the colour keys. See how easy it is to print in colour.

For a balanced colour program, enter and run the following.

```
10 PRINT"■";CHR$((32*RND(1))+96);
20 PRINT"■";CHR$((32*RND(1))+96);
30 PRINT"■";CHR$((32*RND(1))+96);
40 PRINT"■";CHR$((32*RND(1))+96);
50 PRINT"■";CHR$((32*RND(1))+96);
60 PRINT"■";CHR$((32*RND(1))+96),
70 GOTO10
```

Some of them will be terrible patterns, others rather good.

Try changing the comma in line 60 to a ; You'll want to play with this for a long time.

You can make the program a little shorter using the DEF statement. Add LINE 1 as follows:

```
1 DEFFNR(X)=INT((32*RND(X))+96)
```

now whenever you see the phrase INT(32*RND(1)) + 96) in the program (lines 10-60) replace it by the phrase FNR(1). Whenever the computer comes across the word FNR it looks at line one which tells it what it means. We'll be using DEF again.

Building a Library

I hope you've been SAVEing the programs you create as you go along. There is little point in having to input the whole program by hand every time you want to run it.

Resist the temptation to put all your programs on one cassette, one program after another. The frustration you'll experience searching through the tape (even if you've identified each program with a voice label) to find a particular program is just not worth the trouble. Go to your computer shop, or buy by mail from one of the many companies that advertise in the personal computing magazines, and get a set of C-12 cassettes. Put just one program on each cassette, with two copies of each program on each side, just in case something happens to one of the copies and you find it difficult, or impossible, to LOAD.

Write the name of the program on the cassette, and on the cardboard insert. You'll find this makes it very easy to find the program you want, and as your library builds, it will give you quite a feeling of accomplishment to see all those programs ranked side by side. It will impress your friends as well, who will believe after visiting you, and playing with your computer, that you're some kind of natural computer genius (which you probably are).

Review

Let's review the ground we've covered so far;

- The use of the random number generator J = INT(X*RND(1)) + 1
- The output statement PRINT
- The use of, and ; in PRINT statements
- The control statement IF...THEN...
- The use of strings (A$), including the use of a string to stop a program until RETURN is pressed
- The statement GET
- The symbols < and >
- The operator AND
- The statement END
- The statements FOR...TO/NEXT
- The form and use of a "counter" to limit the number of times part of a program is run through
- How to develop a game program from the core of it, to make it more elaborate and interesting
- Subroutines (with the commands GOSUB and RETURN)
- How the sequence FOR...TO.../NEXT, and PRINT can be used to generate pattens.

We've come quite a long way in a short time. Make sure you understand all the preceding material before going on.

For/Next

You'll remember when we wrote the first PATTERN-MAKER program, we used (in line 20) FOR J = 1 TO 75, and (in line 40) NEXT J. We will now have a look at the use of FOR/NEXT loops in some detail.

First, input the following program:

```
10 J=0
20 PRINT"J=";J
30 IFJ=10THENEND
40 J=J+1
50 GOTO20
```

Run it a few times, then clear the memory and input the following program, which uses the FOR/NEXT loop:

```
10 FORJ=1TO10
20 PRINT"J=";J
30 NEXTJ
```

You can see that running this program produces exactly the same result as running the preceding one, although this second program is only three lines long, two shorter than the first program. It is not always possible or desirable to use a FOR/NEXT loop as a "counter", but because it uses less memory than the J = J + 1 approach, it should always be investigated.

When you run a program with a FOR/NEXT loop, the computer goes from line to line after the FOR statement until it finds the NEXT command, when it reverts to the FOR line. You can place one (or more) FOR/NEXT loops inside another.

If line 10 had read FOR J = 1 TO 5 then 5 numbers would have been printed. If it had read FOR J = 1 TO 20 then 20 numbers would have been printed. If it had read FOR J = 1 TO 0 then no numbers at all would be printed, because control would jump straight away to the line after the NEXT J line. Leaving line 10 as it was add the following:

```
22 FORK=1TO2
25 PRINTK
27 NEXTK
```

Run this. It is important to make sure that each loop nests neatly within each other loop. That is, if you placed the NEXT K command after the NEXT J, so the loops cross, you get a rather unsatisfactory result.

Finally, here is a remarkably effective double FOR/NEXT loop program:

```
5 PRINT"■YOUR NAME IN LIGHTS"
10 FORJ=1TO6
20 PRINT,"(PUT YOUR FIRST NAME HERE)";
30 FORS=1TO99
40 PRINT"♠";
50 NEXTS,J
60 END
```

This is a good program to run for your egocentric friends. Note that in line 50 NEXT S,J is the same thing as NEXT S followed by NEXT J.

Flipping a Coin

The computer can be used to simulate a huge variety of things, from the growth of plants to the effects of bumps on the road on motorcycle suspension. We are going to look at a more homely example of simulation: flipping a coin.

Input the following program, and run it:

```
10 PRINT"□   COIN FLIP-1"
20 J=INT(2*RND(1))
30 IFJ=1THENPRINT"HEADS"
40 IFJ=0THENPRINT"TAILS"
50 END
```

Once you've run this a few times, amend the program as follows:

```
10 PRINT"□   COIN FLIP-2"
20 J=INT(2*RND(1))
30 IFJ=1THENPRINT"HEADS"
40 IFJ=0THENPRINT"TAILS"
50 PRINT"TO FLIP AGAIN PRESS F"
60 INPUTA$
70 IFA$="F"THENRUN
80 END
```

You can run this until the screen is full. Although there will probably be an imbalance in the numbers of HEADS and TAILS, the longer you run the program, the more the ratio of each should approach 1:1. If we knew how many times we had flipped the coin, it would make it easier to work out how many HEADS and how many TAILS the program had printed. It is fairly simple to amend the program to do this. Try it yourself, before looking at my version.

```
10 K=1
20 PRINT"□   COIN FLIP-3"
30 J=INT(2*RND(1))
40 PRINT"FLIP NUMBER";K;"IS ";
50 IFJ=1THENPRINT"HEADS"
60 IFJ=0THENPRINT"TAILS"
70 K=K+1
80 PRINT"TO FLIP AGAIN PRESS F"
90 INPUTA$
```

```
100 IFA$="F"THEN20
110 END
```

Once you've run this a few times and added up the HEADS and TAILS, and then worked out what the ratio of HEADS to TAILS is, you will probably have realised the computer can do all the work — flip the coin, count the heads and tails, and then manipulate this result as you choose. Let's try a more complex program:

```
10 H=0
20 T=0
30 K=1
40 PRINT"□   COIN FLIP-4"
50 J=INT(2*RND(1))
60 PRINT"FLIP NUMBER";K;"IS ";
70 IFJ=1THENPRINT"HEADS"
80 IFJ=0THENPRINT"TAILS"
90 IFJ=1THENH=H+1
100 IFJ=0THENT=T+1
110 PRINT"OUT OF";K;"FLIPS YOU    HAVE"
120 PRINTH;"HEADS AND";T;"TAILS"
130 K=K+1
140 PRINT"PRESS F TO FLIP AGAIN"
145 INPUTA$
150 PRINT"□"
160 IFA$="F"THEN40
170 PRINT"THANKS FOR PLAYING.BYE"
180 STOP
```

You will find that adding PRINT statements for lines 105, 106 and 107 will make the program easier to read. Now, we could use a FOR/NEXT loop to tell the computer in advance how many times we wanted to flip the coin, and then we could leave it to do the work. For practice, I'd like you to produce such a program, which also asks the player at the beginning how many times he or she wishes to flip the coin. I am not going to give you a sample version of this because I think by now you should be able to write such a program easily. What I am going to give you though is another version of the program, that gives approximate percentage breakdowns of heads and tails. First write your program using the FOR/NEXT loop, SAVE it, and then have a look at my final COIN FLIP program.

```
10 PRINT"□    FLIP-A-COIN"
20 H=0
30 T=0
40 K=1
```

```
50 J=INT(2*RND(1))
55 PRINT
57 PRINT
60 PRINT"FLIP NUMBER";K;"IS ";
70 IFJ=1THENPRINT"HEADS"
80 IFJ=0THENPRINT"TAILS"
90 IFJ=1THENH=H+1
100 IFJ=0THENT=T+1
105 PRINT
106 PRINT
107 PRINT
110 IFK=1THEN150
111 A=INT(100*H/K+.5)
112 B=INT(100*T/K+.5)
113 IFA+B>100THENA=A-1
120 PRINT"IN";K;"FLIPS YOU HAVE"
130 PRINT"APROX.";B;"% TAILS"
140 PRINT"AND";A;"% HEADS"
142 PRINT
147 PRINT
150 PRINT"PRESS F TO FLIP AGAIN"
155 K=K+1
160 INPUTA$
170 PRINT"□"
175 IFA$="F"THEN50
180 IFK=2THEN195
181 PRINT
182 PRINT
185 PRINT"YOU FLIPPED THE COIN",K-1;"TIMES:-"
186 PRINT
187 PRINT"HAD";A;"% HEADS"
188 PRINT
189 PRINT"AND";B;"% TAILS"
190 PRINT
191 PRINT
192 PRINT
193 PRINT
194 PRINT
195 PRINT"THANKS FOR PLAYING.BYE"
200 END
```

There are few pints you can learn from examining this program. Up to line 100, there is nothing new. Line 110 (IF K = 1 THEN 150) bypasses all the percentage computation, because there is no need for it if you have only flipped once. Lines 111 and 112 multiply the numbers of heads and tails by 100, then divide by the total number of flips.

Line 113 adds the two "percentages" together, and if they equal 101 (which can only happen if 100*H/K happens to be something point five) 1 is subtracted from the HEADS' percentage, just to make the two percentages add up to 100. Line 180 (IF K = 2 THEN 195) is used if the player decides to stop after two flips. The information given by lines 185 to 189 is only of interest if the coin has been tossed three or more times. Line 195 is just a friendly way of ending.

Whenever you have room in the memory, it is a good idea to make the program more "user-friendly" and make the "conversation" as natural as possible.

Number Crunching

The programs we've worked on so far have ignored the great, and fundamental ability the VIC has to work with numbers.

Input this program into your computer:

```
10 PRINT"□MATHS DEMONSTRATION"
15 PRINT
20 INPUT"GIVE ME A NUMBER";A
25 PRINT
30 PRINT"A SECOND NUMBER,PLEASE"
35 INPUTB
40 INPUT"AND ONE MORE";C
45 PRINT
50 PRINT"SO";A
60 PRINT"MULTIPLIED BY";B
70 PRINT"DIVIDED BY";C
80 PRINT"EQUALS";A*B/C
```

Run this through a few times, with different values for A, B and C. Notice how, in lines 50 to 80, when using PRINT, the computer substitutes the values assigned to A, B and C. It prints the number rather than the letter.

Now, change lines 50 to 80 as follows:

```
50 K=(A+C)/(B+C-A)
60 PRINTK
70
80
```

Run this a few times. We can use a program of this type to do practically any maths problem, no matter how complex. We'll use the basic mathematical ability of the computer to produce a rather interesting demonstration program you can run next time you have a couple of friends around.

```
10 PRINT"■NUMBER JUGGLE"
30 INPUT"NAME OF PLAYER";A$
50 PRINT"■";A$;" PLEASE"
70 PRINT"THINK OF A NUMBER AND"
80 INPUT"DOUBLE IT";C$
120 PRINT"■NOW ";A$;",ADD FIVE"
125 INPUTD$
130 PRINT"■THEN ";A$;",SUBTRACT SEVEN"
140 PRINT"AND TYPE IN THE"
145 INPUT"RESULT";A
150 K=(A+2)/2
180 PRINT"■YOUR NUMBER",A$;",WAS";K
```

Now that did not, perhaps, seem very startling, although it is fairly impressive the first time you try it on someone. Note how the strings C$ and D$ were used to stop the program until you pressed RETURN, how CLR was used to clear the screen after each instruction, and how the final line (180) used the name string (A$) and the assigned variable (K) to print out your answer.

Before you read on, work out a way of letting the program give you the chance to have another go. Type your version in at the end of the program, and see if it works. One way would be as follows:

```
200 PRINT"ANOTHER GO ";A$
205 INPUTZ$
210 IFZ$="YES"THEN70
230 END
```

Once you've run through this program a few more times, you're probably pretty bored with it. Remember that a computer can, within the limits of its memory, store a great deal more information than just one simple mathematical manipulation. This game program becomes a great deal more interesting when you interweave a second set of instructions into the first, so a second player can "think of a number" and so on, at the same time as the first person is doing so. I'll list the whole program.

```
10 PRINT"□NUMBER JUGGLE"
20 PRINT
30 PRINT"NAME OF FIRST PLAYER?"
40 INPUTA$
45 PRINT
50 PRINT"NAME OF SECOND PLAYER?"
60 INPUTB$
65 PRINT"□FIRST ";A$
70 PRINT"THINK OF A NUMBER AND"
75 PRINT"DOUBLE IT"
80 PRINT
82 PRINT"AND YOU ";B$
85 PRINT"THINK OF A NUMBER AS  WELL"
90 INPUT"AND ADD SIX TO IT";C$
100 PRINT"□AND ";B$;","
110 PRINT"I WANT YOU TO MULTIPLYYOUR NUMBER BY THREE"
115 PRINT
118 PRINT"AND YOU ";A$
120 INPUT"ADD FIVE";D$
130 PRINT"□ALSO ";A$
135 PRINT"SUBTRACT SEVEN"
140 PRINT"AND TYPE IN THE RESULT"
145 INPUTA
150 PRINT"□";B$
155 PRINT"I WANT YOU TO SUBTRACT"
160 PRINT"FOUR THEN,TYPE IN YOURRESULT"
170 K=(A+2)/2
175 INPUTB
180 L=((B+4)/3)-6
185 PRINT"□YOUR NUMBER,";A$,"WAS";K
190 PRINT
195 PRINT"AND YOURS,";B$,"WAS";L
200 INPUT"ANOTHER GO";Z$
210 IFZ$="YES"THENGOTO65
```

```
215 PRINT"■"
220 PRINT"OK,";A$;" AND ";B$,"THANKS"
225 PRINT"FOR PLAYING,BYE..BYE.."
230 FORI=1TO1000:NEXT
240 GOTO220
```

Try this out with a couple of friends, and watch their reaction. I want to introduce you now to a little trick which can add a lot to games programs (even though it will probably horrify computer purists), The VIC works very, very quickly. So quickly, in fact, that its responses appear to be instantaneous. Although this is fine for "serious" use of a computer, it detracts in some measure from games. You'll find that games are more interesting if the computer appears to be "thinking" between moves, rather than responding as soon as you hit RETURN. So, we can introduce a subroutine which will slow things down a bit.

Add the following:

```
240 FORI=1TO5000
250 NEXTJ
255 PRINT"■"
260 RETURN
```

Now, go through the program, and change all the lines which have the instruction PRINT "*CLR*" to GOSUB 240. Now, run the program again. Notice how much more effective this seems, with just the right delay for the computer to "think and reach a decision". You could have other numbers in line 240. Try it as FOR J = 1 TO 20000. You'll find this delay is far too long. So long that you'll get irritated. Try other numbers, until the delay seems about right. You can use the subroutine delay in any games you like. As an exercise, try adding it to an earlier program, like RUSSIAN ROULETTE, and seeing if this adds to the playing of the game.

The general form of a "game delay":

Line number	FOR X = n, TO n_2
Line number	NEXT X
Line number	PRINT "*CLR*"
Line number	RETURN (if the delay is a subroutine)

As a general rule, you can insert a delay subroutine whenever you would normally clear the screen.

Writing a Program — Some General Thoughts

Most books on programming suggest you start by drawing up a "flowchart" — a pretty combination of circles, diamonds and slanting rectangles — which sets out the path and operations the computer will follow to execute a program. In theory, this is fine. But in practice, the time and trouble involved is probably not worth it.

It is, however, essential to know exactly what you want the computer to do before you start creating a new program, even if you're not quite sure how you are going to get the computer to carry out the task.

Sometimes a rough sort of flowchart — just writing down the main steps the computer will take and then linking these by lines and loops — will help to clarify your thinking. This is also a good way of spotting potential problems, like the danger of setting up infinite loops, or of not specifying the nature of the computer's decisions exactly.

For what it's worth, I work as follows:

Having worked out the general idea for a game (like "player thinks of an animal, computer tries to guess which animal") I then just think about the idea for a while. I might not write anything down for even a day or more. Usually, this "thinking time" allows me to work out roughly how the core of the program is going to look. Next, on paper, I write down this core, starting with line number 100. This ensures there is plenty of room at the start of the program to add a title and instructions, define constants, set up arrays and the like.

The next stage is to feed the rough program into the computer and see if it does what it is meant to do. Often a far more elegant method than the first rough version is found at the keyboard, but this would probably not have been discovered if the written version was not tried first.

If you're working on a fairly simple game, or are adapting from a magazine or book, it is just as well to work directly on the computer, but keep a notebook handy to record things like the fact that, for example, string A$ is for the player's name.

If you're writing a program direct on the keyboard, and you want to add a subroutine which will eventually be at the end of the rest of the program, give it a very big number (like 5000). It can easily be renumbered afterwards (and the GOSUB command changed). This is simpler, and more elegant, as well as using less memory, than having to use a GOTO to jump over a subroutine which has been given too low a line number.

If there is a phrase that you'll need the computer to print at several parts of the program, such as "THE SCORE AT THIS POINT IS", assign a string to this phrase as soon as you realise you're going to have to place the line in several places in the program. A lot less memory is involved in the line PRINT B$ apparing six times, than in six of PRINT "THE SCORE AT THIS POINT IS". If the information to be included several times is longer than one line, a subroutine is probably the most memory-efficient device to use. Later on this book looks at a program which makes use of a large number of strings for phrases which are used over and over again, and when you come to the game, you'll see how efficient a process this can be.

Extra-Sensory Perception

The next program we will look at uses the computer "greater than" and "less than" facilities to compare two numbers, and then makes a decision on the basis of whether one number is less than, or greater than, the other. This program also makes use of many of the things we've covered so far, including the IF ... THEN, the counter and the random number generator.

Input the following program:

```
10 PRINT"□GUESS MY NUMBER"
20 PRINT
45 PRINT"WHAT NUMBER,BETWEEN   1AND 100"
46 PRINT"AM I THINKING OF?"
50 J=INT(100*RND(1))+1
55 S=0
60 S=S+1
70 INPUTA
75 IFJ=ATHEN105
80 IFA<JTHENPRINT"HIGHER"
85 IFA>JTHENPRINT"LOWER"
90 IFS<5THEN60
95 GOTO120
105 PRINT"YOU ARE RIGHT I WAS   THINKING OF ";J
110 GOTO140
120 PRINT"TIME IS UP"       130PRINT"I WAS THINKING OF ";J
140 PRINT"DO YOU WANT ANOTHER GO"
145 INPUTH$
150 PRINT"□"
155 IFH$="YES"THEN45
160 END
```

Notice how the core of the program (lines 50 to 95) make the decision on what the computer is to do next.

It is possible to dress the program up a bit, by giving the computer your name, and — as you'll see — giving it the ability to award the player a random payoff if the number is guessed correctly. Try this new form. Some of the original program remains, but certainly lines have to be replaced, and others added.

```
10 PRINT"□GUESS MY NUMBER"
20 PRINT
25 PRINT
30 PRINT"HI,WHAT IS YOUR NAME?"
35 INPUTA$
40 PRINT"□"
45 PRINT"WHAT NUMBER, BETWEEN 1AND 100,AM"
46 PRINT"I THINKING OF,";A$;"?"
50 J=INT(100*RND(1))+1
55 S=0
60 S=S+1
70 INPUTA
75 IFJ=ATHEN100
80 IFA<JTHENPRINT"HIGHER"
```

```
85 IFA>JTHENPRINT"LOWER"
90 IFS<5THEN60
95 GOTO130
100 K=INT(3*RND(1))
105 PRINT"YES I WAS THINKING OF ";J;" I HEREBY
          AWARD YOU THE TITLE OF";
110 IFK=0THENPRINT"SMART ALEC OF THE YEAR,";A$
115 IFK=1THENPRINT"MASTER OF E.S.P.";A$
120 IFK=2THENPRINT"JERK OF THE KEYBOARD ";A$
125 GOTO140
130 PRINT"YOU ARE STUPID,";A$,"I WAS"
135 PRINT"THINKING OF";J
140 PRINT
145 PRINT"DO YOU WANT ANOTHER GO"
150 INPUTN$
155 PRINT"□"
160 IFN$="YES"THEN45
165 PRINT"THANKS FOR PLAYING ";A$
170 END
```

When you run this program, you'll find the lines 105 to 120 may wrap around to the following line when printing. This is not a good feature. As an exercise, try and rewrite this section so the lines print separately, so you do not run the risk of having a word cut in half, with one half on one line, and the next on the other.

Kill the Chopper

It is possible to use a very similar core to produce what appears to be a completely different program. Input the following:

```
10 PRINT"□CHOPPER"
20 PRINT
30 PRINT
40 PRINT"A DEADLY CHOPPER IS   HIDING"
45 PRINT"1 OF 16 TREES"
50 PRINT
55 PRINT
60 INPUT"YOU ONLY HAVE 6 SHOTS,OK";A$
70 PRINT"□"
75 G=0
80 J=INT(16*RND(1))+1
85 PRINT"●    ●"
90 PRINT"┘ ♠ ♠ ┘"
95 PRINT"┘ < > ┘"
100 PRINT"===="
105 PRINT"● ●"
115 PRINT
120 PRINT
125 PRINT"GUESS THE NUMBER OF   THE TREE"
135 PRINT"YOU WANT TO SHOOT AT"
140 G=G+1
145 INPUTM
150 PRINT"□"
155 IFM=JTHEN220
160 IFG=6THEN195
165 PRINT"YOU MISSED,";6-G;"TO GO"
170 PRINT
175 PRINT
180 IFM<JTHENPRINT"TRY TO THE RIGHT"
185 IFM>JTHENPRINT"TRY TO THE LEFT"
190 IFG<6THEN85
195 PRINT"AAAAAAAAARGH....."
200 PRINT
205 PRINT
210 PRINT"HE WAS BEHIND TREE    NUMBER";J
215 END
```

```
220 PRINT,"GOT HIM"
225 PRINT"┘ ●● <>"
229 PRINT,"GOT HIM"
230 PRINT" ┘==┘= ●"
235 PRINT
240 PRINT
245 PRINT
250 PRINT"PRESS 2 FOR ANOTHER GO"
255 PRINT"OR 4 TO CHICKEN OUT"
260 GETT$
265 IFT$="2"THEN70
270 IFT$<>"4"THEN260
275 PRINT"■","CHICKEN"
280 END
```

The idiotic appearance of the "chopper" was chosen at random. (It was named after my dog.) Make up your own name and appearance for your beastie (making sure its "dead" version looks vaguely like a deflated version of the live creature). You can also alter the value of J (the number of trees) and/or G (the number of shots you get before being killed).

This framework can also be used for, say, vultures hiding in nests, poisonous snakes in holes, or even unfriendly aliens in space, lurking behind asteroids. The "conversation" parts of the program can also be altered to what ever form you choose.

A little gimmick you can add to this (and any other program you like assuming you have enough money for it) is to get the program to ask the player's name. You work out a subroutine which ensures the computer will only play with you (or with people fortunate enough to share your name). If any other name is fed in when the computer asks who is playing, it goes straight to END. As an exercise, work out such a subroutine and add it to your version of CHOPPER.

Now you've probably thought you could produce a better looking beastie than the CHOPPER of mine. Enter and run the following program, to see just how many graphical alternatives you have:

```
10 FORJ=96TO127
20 PRINTJ;" ";CHR$(J),J+64;" ";CHR$(J+64)
30 FORN=1TO2000:NEXT
40 NEXT
```

You can slow the program down even more by holding down the CTRL key while running.

As well as seeing the different symbols you have at your disposal (all created by PRINT CHR$(X)) you can learn from the above program two things:

1) In line 30, you can join two lines together with a colon; and
2) You do not need to add a variable name after NEXT (lines 30 and 40)

Now, re-write CHOPPER using your own choice of graphics to create a better looking deadly monster.

Days of Our Lives

Let's go back now to number-crunching, and have a look at a program which gives interesting (and sometimes scary or absurd) information on how long the player has to live. A little later on in this book is a program designed to give a proper look at life expectacny, but that program is a little more complicated for now. This program — DAY OF OUR LIVES — produces a pretty good demonstration of your computer, and the less mathematical of your friends will be convinced, yet again, that you are some kind of genius.

What the program does, in essence, is work out approximately how old the player is in days, and compares this with average life expectancy (67 ½ years for males, 74 ½ for females). It also assumes the player sleeps eight hours a night. Input the following program and run through it a few times. Then read the discussion which follows the program listing.

```
10 PRINT"□DAYS OF OUR LIVES"
20 PRINT
30 PRINT"HI WHAT IS YOUR NAME"
35 INPUTA$
40 PRINT"□OK,LETS WORK OUT SOME"
50 PRINT"INTERESTING THINGS    ABOUT YOU ";A$
55 INPUTB$
60 PRINT"□HOW OLD ARE YOU IN    YEARS"
65 INPUTA
70 INPUT"AND MONTHS";B
80 X=365*A+30*B
90 PRINT"□YOU ARE ";X
95 PRINT"DAYS OLD ";A$
100 PRINT"ARE YOU FEMALE"
105 INPUTD$
```

```
110 PRINT"□"
115 IFD$="YES"THEN130
120 Y=24637-X
125 GOTO135
130 Y=27192-X
135 PRINTA$;",YOU HAVE"
136 PRINTY;" DAYS"
138 PRINT"TO LIVE-BASED ON      STATISTICS"
140 INPUTE$
150 PRINT"□YOU HAVE SLEPT FOR    ABOUT";INT(X/3)
160 INPUT"DAYS SO FAR";F$
170 PRINT"□AND WILL SLEEP FOR    ABOUT";INT(Y/21)
175 PRINT"WEEKS OF YOUR"," REMAINING LIFE"
180 INPUTG$
190 PRINT"□IS THERE ANYONE ELSE  THERE WHO"
195 PRINT"WOULD LIKE A GO,"
198 PRINTA$
200 INPUTH$
210 IFH$="YES"THENRUN
215 PRINT"OK BYE.";A$
220 END
```

If you were thinking about what you were entering into the computer, you probably have a pretty good idea of what each line and command was for, but, if not, look at it on your screen and follow through this next section, line by line with your program.

First of all, of course, the program got your name (A$), then stopped at line 50 waiting for you to press RETURN in order for the program to continue (and clear the screen at line 60). This stopping of the program, as you learned earlier, does not do anything, but makes the program much more interesting to run than supplying all the information at once.

The program obtained your age in years (A) and months (B), then worked out what this was in days (line 80, which assumed there were 30 days in a month). Next, in line 90, the computer told you what this figure was (X). The question in line 100 — ARE YOU FEMALE? — is needed because males and females have different life expectancies. If D$ (the answer to the line 100 question) is YES, the computer goes to line 130, where it subtracts your age in days from the life expectancy (74½ years converted to days) for females born after 1961 (it is only half a year less for females born 1951-1960). If the answer is not YES, the computer moves to the next line (120) and subtracts your age in days from 24,637 days, the life expectancy for males born after 1960 (again, it is half a year less for males born 1951-1960).

Line 135 prints how many days after are left. Lines 150 and 160 give the number of days slept so far (your age in days, divided by three), assuming you sleep eight hours a night. Then the computer takes your remaining expected days (Y), divides this figure by three (to get the days spent in sleep) and then divides this by seven to convert this figure to weeks. Actually, as you can see, the program simply divides by 21 (ie 3 x 7).

Lines 190 to 210 ask if someone else wants a go, and if so, goes right back to line 30, where the string A$ is ready to be assigned to the new name. If there is no-one else to play the computer proceeds to the end of the program.

If you feel like experimenting, there is no reason to stick with the program as I've written it. Let it ask, for example, if the player is male (changes lines 100 to 130). Instead of the somewhat unimaginative farwell (line 215) you might wish to put in some mournful line like:

WELL, A$, I GUESS YOU'D BETTER RUN OFF AND MAKE THE MOST OF THE Y/30 MONTHS YOU HAVE LEFT TO LIVE.

You could also, of course, add a delay subroutine everytime CLR appears in the program. By all means experiment with the program, and put your own personal stamp on it. By doing this, you'll gain far more knowledge about programming than you will just by feeding in the programs in this book, line for line, and then leaving them this way.

If you'd like to work out a whole new program based on the core of this program, feed the following program into your computer, and then work around it as you choose.

```
10 PRINT"■HOW OLD ARE YOU IN    YEARS"
15 INPUTA
20 INPUT"AND MONTHS";B
30 X=365*A+30*B
40 INPUT"ARE YOU FEMALE";A$
50 IFA$="YES"THEN80
60 Y=24637-X
70 GOTO90
80 Y=27192-X
90 PRINT"■YOU ARE";X,"DAYS OLD AND"
100 PRINT"HAVE ABOUT";Y,"DAYS TO LIVE,"
110 PRINT"BASED ON STATISTICS"
120 PRINT"YOU HAVE SLEPT FOR","ABOUT";INT(X/3)
130 PRINT"DAYS SO FAR.","AND WILL SLEEP"
140 PRINT"FOR ABOUT";INT(Y/21),"WEEKS OF"
150 PRINT"YOUR REMAINING LIFE"
```

Toying with the Muse

To make a change from working with numbers, here is a program that — after a fashion — writes poetry. Most of the poems it comes up with are pretty awful, but from time to time you'll get a real gem. And even the worst efforts of the computer are not as bad as some modern poetry published today.

The heart of this program is our old friend the random number generator. Input the program as it appears below, and run it several times. Then read the notes that follow the listing. (Note that there is a space just inside the quote marks in the "seed-phrase" lines following the command PRINT.)

```
10 INPUT"□SEASIDE POETRY";A$
30 PRINT"□"
35 FORI=1TO10+INT(11*RND(1))
40 J=INT(21*RND(1))+1
50 IFJ<5THENPRINT:PRINT
60 IFJ=5THENPRINT"...";
70 IFJ=6THENPRINT"SUN"
80 IFJ=7THENPRINT"SAND"
90 IFJ=8THENPRINT"SEAGULLS"
100 IFJ=9THENPRINT"WAVES ";
110 IFJ=10THENPRINT"ROCKS ";
120 IFJ=11THENPRINT"SEAWEED ";
130 IFJ=12THENPRINT"HOT ";
140 IFJ=13THENPRINT"GRITTY ";
150 IFJ=14THENPRINT"BEATING,BEATING ";
160 IFJ=15THENPRINT"HARSH ";
170 IFJ=16THENPRINT"HOURS PASSING ";
180 IFJ=17THENPRINT"ECHOED OVER ";
190 IFJ=18THENPRINT"BRIGHTLY DREW ";
200 IFJ=19THENPRINT"DREAMLY EASED ";
210 IFJ=20THENPRINT"SHADOWED OVER ";
220 IFJ=21THENPRINT"YET AGAIN ";
230 NEXT
```

Note you can use ? instead of PRINT.

When you run this, you'll be quite pleased (or at least I was when I first wrote it) to find out how often, just by chance, quite attractive poems(!) are written by the program. The decisions on where to place the semicolons (;), which tell the computer to print the next words on the

same line, were made partly by running the program, and just seeing how the words fell, and also by the general decision that half the nouns (such a SUN, SAND and SEAGULLS) would end a sentence, that is, would not be followed by a semicolon. I also decided that adjectives (as in lines 130 and 160) would always allow for a word to follow them, and that about half the other "seeds" (lines 170 to 220) would do the same.

By looking at the program listing you'll see the lines 50 to 220 make decisions, based on the random number generated. Line 60 ensures that, just under 5% of the time, "..." will be printed. Line 20 prints two blank lines, just under 15% of the time, and the other lines direct the hardworking computer to print one of the "seeds" just over 80% of the time. The proportions were worked out by running the program over and over again and adjusting the lines until — the majority of the time — a pleasant poem layout resulted.

The kind of poem the program writes depends, as you can see, almost entirely on the words placed in the PRINT lines. The best way to decide on the words to go in the PRINT statements is to pick one topic, and then make every word and phrase relate to this topic.

The original POETRY program was written when I was on holiday, and after producing about 20 SEASIDE POEMS, I decided to input some words and phrases about the city where the program was written. The changes in the program produced some remarkably effective poems (plus a generous crop of duds). If you want to see how it works, change the lines using the following words:

```
10 "SALZBURG POETRY"
80 "BAROQUE TOWERS ";
90 "MOUNTAIN VISTAS ";
100 "MUSIC BY MOZART ";
110 "FORTRESS ";
120 "COBBLED STREETS ";
130 "TIMELESS ";
140 "BELOVED ";
150 "MEMORIES IN STONE ";
160 "STEADFAST ";
170 "TRADITIONS RELIVED ";
180 "ECHOED ";
190 "COPPER DOMES ";
200 "DREAMING ";
210 "SHADOWED OVER ";
220 "FAINTLY WHISPERED ";
```

Try writing a program, using words and phrases based on the last place you spent a holiday in, or pick a topic like "clouds", "kittens" or "vintage cars" and see what you, the computer and the Muse can create. Here is part of one poem written from a set of words and phrases about London.

...TOURISTS CROWD ALWAYS MOVING
RUSHING,PUSHING I HAVE SEEN IT
BIG BEN CHIMES,AND RIVER THAMES
CATHEDRAL SPIRES
ALL BUT FORGOTTEN RUSHING, PUSHING
PIGEONS IN TRAFALGAR SQUARE
I HAVE SEEN IT
AT LAST, SUN, RIVER THAMES
TIMELESS
I HAVE SEEN IT

Not very brilliant, I guess, but acceptable. The "seeds" for this poem are:

```
10 "POEMS OF LONDON TOWN"
20 "BOBBIES "
30 "TOURISTS CROWD ";
40 "PIGEONS IN TRAFALGER ";
50 "CHAOS"
60 "AT LAST,SUN ";
70 "STREETS OF... ";
80 "TIMELESS"
90 "ALL BUT FORGOTTEN ";
100 "MEMORIES OF MAJESTY,"
110 "RIVER THAMES ";
120 "BIG BEN CHIMES,AND ";
130 "RUSHING,PUSHING ";
140 "ALWAYS MOVING,"
150 "CATHEDRAL SPIRES"
160 "GRAY RAIN FALLS ON...";
170 "I HAVE SEEN IT"
180 "MANY TIMES"
```

Notice how, in this program, link words like AND and ON finish some lines.

Here is part of another poem from yet another set of words:

GRAVEYARDS ABOMINATIONS RELIVED
DARK,DARK SHRINGKING WITCHES CACKLE

ECHOED CHILL OF ODDNESS CHANCES LOST
SHRINKING CALLING,CALLING,GRAVEYARDS
ECHOED CALLING,CALLING
DEATH IS NEAR,SAY WITCHES CACKLE
TURN AND REACH FOR
DARK,DARK WITCHES CACKLE
SKELETONS RATTLE
GRAVEYARDS

The "seeds" are as follows:

```
10 "BLACK SABBATH POETRY"
20 "GRAVEYARDS ";
30 "SKELETONS RATTLE"
40 "CHILL OF ODNESS ";
50 "WITCHES CACKLE"
60 "HOPE NOW DEAD ";
70 "CHANCES LOST ";
80 "TIMELESS"
90 "DARK,DARK ";
100 "MEMORIES OF PAIN,"
110 "SHRINKING ";
120 "ABOMINATIONS RELIVED ";
130 "ECHOED ";
140 "CALLING,CALING,";
150 "DEATH IS NEAR,SAY ";
160 "SCREAMS ARE ";
170 "TURN AND REACH FOR ";
180 "AGAIN,AGAIN"
```

Hunting on a Grid

There are many games, with names like HUNT THE HURKLE or BURIED TREASURE or DEPTH-CHARGE, which are based on trying to guess the location of one or more points on a grid. Each location is specified by quoting the co-ordinates of the point. To start our investigation of these games, input the following program, which sets up a very simple game of this type.

```
10 PRINT"■HUNT GAME"
20 X=INT(10*RND(1))+1
30 Y=INT(10*RND(1))+1
35 PRINT
40 PRINT"A FROG IS HIDDEN ON A 10 * 10 GRID"
50 PRINT"YOU HAVE 10 GUESSES TOFIND IT"
60 PRINT"INPUT YOUR GUESS","PRESSING RETURN"
70 PRINT"AFTER EACH DIGIT,","TWO DIGITS ARE"
80 PRINT"NEEDED,THE FIRST ONE  YOU INPUT"
90 PRINT"MUST BE FOR THE","HOROZONTAL CO-ORDINATE"
100 FORJ=1TO10
110 INPUT"FIRST NUMBER";A
140 INPUT"SECOND NUMBER";B
160 IFA=XANDB=YTHEN240
190 PRINT,"TRY AGAIN"
200 NEXTJ
210 PRINT"END OF GAME"
220 PRINT"FROG HIDDEN AT";X;",";Y
230 END
240 PRINT"YOU HAVE FOUND IT":PRINT:PRINT
245 FORI=1TO1000
248 NEXT
250 GOTO240
```

Once you've played this a few times, you'll realise that you're really "shooting in the dark" when trying to guess the frog's location, and there is no feedback as to how close you are.

You can add the following lines to give you an idea of how you're going:

```
170 IFA=XANDB<>YTHENPRINTA;"IS RIGHT,";B;"IS NOT"
180 IFA<>XANDB=YTHENPRINTA;"IS WRONG,";B;"IS RIGHT"
```

Add these lines, and run the program a few times. You can then add another line to give the player more information:

```
175 IFJ=6THENPRINT"CLUE:DIGITS ADDDED=";X+Y
```

Of course, many other features can be added. Here's a version of the hunt game which allows the player to select the size of the grid, and gives clues in terms of direction.

```
10 PRINT"□HUNT THE SPIDER"
20 PRINT
30 PRINT"IN THIS GAME A SPIDER WILL WEAVE"
40 PRINT"A WEB AND HIDE ON IT":PRINT
50 INPUT"HEIGHT OF WEB";X
70 INPUT"AND WIDTH";Y
80 PRINT"□"
90 A=INT(X*RND(1))+1
100 B=INT(Y*RND(1))+1
110 FORJ=1TO5+INT(X*Y/5)
120 PRINT"THE SPIDER IS HIDING  ON A"
130 PRINTX;"BY";Y;"WEB.WHERE","IS IT?"
140 PRINT:INPUT"VERTICAL";C
150 INPUT"HORIZONTAL";D
156 PRINT"□"
160 IFC=AANDD=BTHEN340
165 PRINT"GUESS NUMBER";J,"WAS";C;",";D;"AND WAS"
170 PRINT"WRONG.TRY TO THE"
180 IFC>ATHENPRINT"NORTH ";
190 IFC<ATHENPRINT"SOUTH ";
195 IFD<BTHENPRINT"EAST"
200 IFD>BTHENPRINT"WEST"
210 PRINT
220 PRINT
240 NEXTJ
250 PRINT"GAME OVER",,"SPIDER WAS AT ";A;",";B
280 INPUT"ANOTHER GAME";A$
300 PRINT"□"
310 IFA$="YES"THENRUN
320 PRINT"THANKS FOR PLAYING"
330 END
340 PRINT"YOU FOUND THE SPIDER  IN";J;"TRIES"
350 GOTO280
```

Slot/Fruit Machines

Once you've paid for your computer, bought some software (and this book) and possibly invested in some additional memory, your wallet may be getting a little light. Here's a program which you can use to get a little money out of your richer friends.

The random number generator (of course) can easily be used to simulate a slot machine, or fruit machine, game. A very simple program of this type, with two "fruits" could be as follows:

```
10 PRINT"□SLOT MACHINE"
20 C=5
30 A=0
40 B=0
50 FORJ=1TO2
60 D=INT(2*RND(1))
70 IFD=0THEN110
80 A=A+1
90 PRINT,"ORANGE"
100 GOTO130
110 B=B+1
120 PRINT,"CHERRY"
130 NEXTJ
140 IFA=2ORA=0THENC=C+2
150 IFA=1THENC=C-1
160 IFC<1THEN230
170 IFC>10THEN250
180 PRINT"YOU NOW HAVE $";C
190 PRINT"PRESS RETURN FOR NEXT GO"
200 INPUTE$
210 PRINT"□"
220 GOTO30
230 PRINT"YOU ARE BUST":PRINT
235 FORI=1TO1000
238 NEXT
240 GOTO230
250 PRINT"YOU HAVE BROKEN THE   BANK":PRINT
255 FORI=1TO1000
258 NEXT
260 GOTO250
```

Input this program and run it a few times. It works as follows: Line 20 sets the counter for ORANGEs(A) at 0, and line 30 sets the CHERRY counter (B) at 0. Your money is C, and line 20 sets it at $5 to begin the game.

Each play of the game requires two random numbers to be generated. These are counted by line 50(and 130) and generated by line 60. If the random number is 0, program control moves to line 110 where B become B + 1 and CHERRY is printed. Control then goes back to line 50. If the random number is 1, A becomes A + 1, and the computer prints ORANGE before going back to line 50 for the next number.

After two numbers have been generated, and the "fruits" printed out, the computer checks to see if both are the same (if they are, A will equal 2 or 0) and if so, adds two to your money (that is, lets C = C + 2). If the fruits are not the same, the computer takes $1 off you (that is, lets C = C − 1). Can you see how the computer knows the two fruits were different?

Next, it prints out your stake (line 160). The computer checks this amount. If it is less than £1, control goes to line 230 to print YOU HAVE BUSTED. If it is greater than 10 control moves to line 250 where the computer prints YOU HAVE BROKEN THE BANK. If neither of these conditions are true, the computer acts on the next line (190) which instructs you to press RETURN for the next go. That may seem a little complicated spelt out in detail, but you should be able to follow it through on the program.

The splendid result of breaking the bank is well worth trying to win.

Once you're sure you understand the workings of this program, write your own version with three fruits, and thus make it more like a "real" slot machine. You'll have to add some lines between others in my program to do this, and change a few lines completely. I'm not going to give you a program for a "three fruits" machine as you will benefit far more from working out your own program to simulate the working of such a machine. And it is far more fun playing a game with a program you've written yourself, than just feeding in someone else's work.

Once you've written and played with the three fruits version, read on to see one way of writing a four fruits slot machine (but with pretty odd fruit). Note that in this program, different winning combinations are worth different amounts, and generate comments.

```
5 POKE36879,27
10 PRINT"■SLOT MACHINE"
30 PRINT
40 PRINT
50 PRINT
60 PRINT
70 INPUT"STARTING STAKE(<20)";M
90 PRINT"■"
100 C=0
110 D=0
```

```
120 E=0
130 F=0
140 FORJ=1TO4
150 ON INT(4*RND(1)) GOTO 220,250,280
190 C=C+1
200 PRINT"■COMMODORE"
210 GOTO300
220 D=D+1
230 PRINT"■VIC"
240 GOTO300
250 E=E+1
260 PRINT"■20"
270 GOTO300
280 F=F+1
290 PRINT"■COMPUTER"
300 NEXTJ
310 IFC=1ANDD=1ANDE=1ANDF=1THEN370
320 IFC=4ORD=4ORE=4ORF=4THEN370
330 IFC=3ORD=3ORE=3ORF=3THEN390
350 M=M-2
360 GOTO420
370 M=M+10
375 PRINT"■JACKPOT"
380 GOTO420
390 M=M+2
400 PRINT"3 OF A KIND"
420 IFM<1THEN490
430 IFM>49THEN540
432 PRINT
435 PRINT
437 PRINT
440 PRINT"■YOU NOW HAVE £";M
445 PRINT
450 INPUT"PRESS RETURN FOR NEXT SPIN";A$
480 GOTO90
490 PRINT"END OF GAME.YOU ARE   BROKE"
500 INPUT"ANOTHER GAME";B$
520 IFB$="YES"THEN60
525 PRINT"■OK,BYE BYE"
530 END
540 PRINT"YOU HAVE £";M",AND"
550 PRINT"HAVE BROKEN THE BANK":PRINT
560 GOTO500
```

Lost in Space

We'll return now to the HUNTING ON A GRID idea, and work on finding objects which are located on more than two axes, that is, are hidden in three (or even four) dimensional space. Imagine our spider is hiding inside a cube, each side of which is X units long. Each point in the cube can be specified by giving three co-ordinates: height, width and depth. Before reading on to see how I've done it, try to work out a program in which a space capsule is lost inside a cube of space, with each side of the cube seven kilometres long.

```
10 PRINT"□    LOST IN SPACE"
15 PRINT
20 PRINT"YOU HAVE 15 HOURS TO  FIND A CAPSULE"
30 PRINT"LOST IN A 7KM CUBE"
40 PRINT,"OF SPACE"
50 A=INT(7*RND(1))+1
60 B=INT(7*RND(1))+1
70 C=INT(7*RND(1))+1
80 FORJ=1TO15
85 PRINT
90 PRINT"INPUT 3 SEARCH","CO-ORDINATES"
100 INPUTD,E,F
110 PRINT"□"
130 IFA=DANDB=EANDC=FTHEN320
140 PRINT
150 PRINT"WRONG":PRINT
160 PRINT"HOURS OF AIR LEFT:";15-J
170 PRINT
180 PRINT"MOVE:"
190 IFA>DTHENPRINT"UP ";
200 IFA<DTHENPRINT"DOWN ";
210 IFB<>ETHENPRINT"ACROSS ";
220 IFC<FTHENPRINT"FORWARDS"
230 IFC>FTHENPRINT"BACKWARDS"
240 PRINT
250 IFJ=14THENPRINT"DANGER-DEATH IMMINENT"
260 NEXTJ
280 PRINT"□"
290 PRINT"FAILURE-",,"ASTRONAUTS DEAD":PRINT
300 PRINT"CAPSULE WAS AT ",A;B;C
310 GOTO350
```

```
320 PRINT
330 PRINT"YOU FOUND THEM WITH"
340 PRINT16-J;"HOURS OF AIR LEFT"
350 PRINT
360 INPUT"ANOTHER MISSION";H$
380 PRINT"J"
390 IFH$="YES"THEN20
400 PRINT"FAREWELL,BRAVE CAPTAIN";
410 GOTO400
```

Arrays

Arrays, and the use of the DIM (dimension) statement puzzle many newcomers to BASIC. The DIM is used if you want to set up a list with a 'name' (the name is a letter like A, B or whatever). For example, you might want the number 1 to 15 in the list named A, in the form LET A(Ø) = 1, LET A (1) = 2 and so on to LET A(14) = 15. To tell the computer you need an array to hold these elements (1,2 and so on to 15) input:

10 DIM A(14)

The figure in brackets can be one less than the number of items or elements you want in the list because, in BASIC, an array contains one more element than the number you place in the brackets; the first element is A(Ø), not A(1). However, you can have a much bigger array if you like, provided you don't exceed the memory. In practice, it is sufficient to put

the number of elements you want in the brackets, 'forgetting' that you are actually creating an array one element bigger than you need. The number inside the brackets is known as the subscript, and an element of the form F(2) or K(9) is called a subscripted variable.

You can also have multidimensional arrays. This means arrays with more than one subscript. Again you use a DIM statement to set them up. For example

```
DIM B(2,2)
```

creates a two dimensional array (the dimension is the number of subscripts) which has nine elements. These elements are called B(Ø,Ø), B(Ø,1), B(Ø,2), B(1,Ø), B(1,1), B(1,2), B(2,Ø), B(2,1) and B(2,2).

When using large arrays it is often better, in terms of memory space, to use an integer array. eg DIM B%(2,2) which creates an array of nine integer elements.

You can also have string arrays. DIM AB$(3,1) sets up a string array called AB$ with eight ((3 + 1) x (1 + 1)) elements.

Here's a program to show the DIM statement:

```
1Ø DIM A(1Ø)
2Ø FORJ=ØTO1Ø
3Ø A(J)=2*J
4Ø PRINT"A(";J;")=";A(J)
5Ø NEXTJ
```

When you run this, you will get:

A(Ø) = *Ø*
A(1) = *2*... and so on to ...
A(1Ø) = *2Ø*

Change line 1Ø to A(5) and run the program again. This time you'll get just A(Ø) = Ø... A(5) = 1Ø. The program stopped at this point (giving an error message) meaning that the sub-scripted variable required, ie J = 6 to J = 1Ø was out of range. Change line 1Ø now to DIM A(2Ø) and run the program. You'll find you get exactly the same result as having DIM A(10). As I said before, you change nothing, in practice, by having a larger array than you actually want. Now add the following lines:

```
1Ø DIM A(1Ø)
6Ø INPUTB$
```

```
70 FORZ=0TO15
80 LET A(Z)=3*Z
90 PRINT"A(";Z;")=";A(Z)
100 NEXTZ
```

When you run this, you find the same A(0) = 0 through to A(10) = 20 followed by the symbol asking you to input a string (or, as in this case, press RETURN). The computer will then display:

A(0) = 0
A(1) = 3
...down to ...*A(10) = 30*

The error code will be displayed, because the demand made on the array by line 80 was greater than the array defined by line 10 (that is, line 70 made the next Z equal 11, and line 80 therefore wanted a subscripted variable called A(11) which, of course, it could not find because the array only had room for 11 elements).

Type NEW and input the following program, which uses string arrays:

```
10 DIMA$(10)
20 FORA=1TO10
30 INPUTA$(A)
40 NEXTA
50 PRINT"□"
60 FORA=1TO10
70 PRINT"A$(";A;") IS ";A$(A)
80 NEXTA
```

As you can see, you can have many different strings, each of which may be of a different length.

Now follows a very simple HUNT THE HURKLE program using the DIM statement. Input the program, run it a few times, then read through to find out how it works.

```
10 PRINT"DIM SPIDER"
20 DIMA(4)
30 DIMB(4)
50 FORG=1TO4
60 A(G)=2
70 B(G)=2
80 NEXTG
```

```
90 K=INT(4*RND(1))+1
100 L=INT(4*RND(1))+1
110 A(K)=1
120 B(L)=1
130 FORD=1TO5
140 PRINT"WHERE IS SPIDER,TRY   NUMBER";D
150 INPUTT,Y
170 IFA(T)=1ANDB(Y)=1THENPRINT"YOU FOUND IT"
180 IFA(T)=1ANDB(Y)=1THENEND
190 NEXTD
200 PRINT"SORRY,TIME IS UP"
210 PRINT"SPIDER WAS AT",K;L
```

Having run this program a few times, and examined the listing, you are probably asking yourself what was achieved by using the DIM statements which could not have been achieved without them. The answer is: Nothing. However, the DIM comes into its own when you want to hide more than one object on a grid, without creating a whole set of co-ordinates of the type A = INT(4*RND(1)) + 1, B = INT(4*RND(1)) + 1, C = INT(4*RND(1)) + 1 and so on, to put the first hidden object at A,B; the second at C,D; and so on.

Before we look at this, here are two more programs which hide a single object.

```
5 PRINT"□    DIMMER SPIDER"
10 DIMA(2)
20 DIMB(2)
30 FORC=1TO2
40 A(C)=INT(4*RND(1))+1
50 NEXTC
60 FORJ=1TO7
70 PRINT"WHERE IS THE SPIDER-  ATTEMPT";J
80 FORC=1TO2
90 INPUTB(C)
100 NEXTC
110 PRINTB(1),B(2);
120 IFA(1)=B(1)ANDA(2)=B(2)THEN200
130 IFA(1)=B(1)THEN220
140 IFA(2)=B(2)THEN240
150 PRINT"NO"
```

```
160 IFJ=5THENGOSUB260
170 NEXTJ
180 PRINT"TIME IS UP.SPIDER WAS AT";A(1);A(2)
190 GOTO205
200 PRINT"YOU HAVE FOUND IT"
210 END
220 PRINT,A(1);"IS RIGHT"
230 GOTO170
240 PRINT,A(2);"IS RIGHT"
250 GOTO170
260 PRINT"HINT:LOCATIONS ADD UP TO";A(1)+A(2)
270 RETURN
```

If you like, you can change line 160 to read:

```
160 IFJ=3ORJ=7THENGOSUB260
```

This just reinforces the same hint when J = 7 as when J = 3. If you want the 'hint' to come at random, you could change line 160 to:

```
160 IFJ=2*A(1)THENGOSUB260
```

or to:

```
160 IFJ=2*A(2)THENGOSUB260
```

Another version of this line is:

```
160 IFJ=7*RND(1)<JTHENGOSUB260
```

This final version is probably the best, because it means that the closer to J = 7 you get, the more likely you are to be given a hint.

The core of the program DIMMER SPIDER can be used to produce a far more interesting game, and in this program we will introduce a new BASIC function:

```
10 DIMA(2),B(2)
20 PRINT"■ESKY PIKSY"
30 FORC=1TO2
40 A(C)=INT(7*RND(1))+1
50 NEXTC
60 FORJ=1TO11
```

```
70 PRINT"WHERE IS PIKSY-ATTEMPT";J
80 FORC=1TO2
90 INPUTB(C)
100 NEXTC
110 PRINTB(1);B(2)
120 IFA(1)=B(1)ANDA(2)=B(2)THEN200
130 IFA(1)=B(1)THEN240
140 IFA(2)=B(2)THEN250
150 PRINT"NO"
160 IF10*RND(1)<JTHENGOSUB260
170 NEXTJ
180 PRINT"TIME IS UP.":PRINT"PIKSY WAS AT";A(1);A(2)
190 END
200 PRINT"□"
205 FORY=1TO5
210 PRINT
220 NEXTY
225 PRINT"YAY, CAPTURED"
230 END
240 PRINTA(1);"IS RIGHT":GOTO170
250 PRINTA(2);"IS RIGHT":GOTO170
260 G=INT(5*RND(1))
270 IFG=0THENPRINT"HINT:LOCATIONS ADD UP TO";A(1)+A(2)
280 IFG=1THENPRINT"HINT:DIFFERENCE
    BETWEEN LOCATIONS IS";ABS(A(1)-A(2))
300 IFG<2THENRETURN
320 PRINT"I WAS AT";A(1);A(2)
330 PRINT"I AM MOVING"
340 PRINT"PRESS RETURN"
350 INPUTA$
370 RUN
```

Have a look at line 280. The function ABS stands for "absolute". If a number is positive the absolute value of that number is just the number. If a number is negative, the absolute value is the number multiplied by −1 (ie the number without its negative sign). If you leave out ABS in line 280, the clue will be so specific it will almost certainly give the location away, so ABS makes for a better game. Lines 200 to 220 perform an interesting task. After clearing the screen (line 200) the FOR/NEXT loop moves the letters to be printed (line 225) to nearer the middle of the screen. This makes for a more attractive display.

The next program locates a number of objects (10) on a grid, and gives a score based, of course, on how many you hit. However, if more than one object is at the same location you get more than one score. The program also awards you a "rating" at the end.

```
10 PRINT"■ALIENS + ASTEROIDS"
20 LETK=0
30 DIMA(10),B(10),C(10),D(10)
100 FORJ=1TO10
110 A(J)=INT(4*RND(1))+1
120 B(J)=INT(4*RND(1))+1
130 NEXTJ
160 FORZ=1TO8
170 PRINTCHR$(INT(32*RND(1))+96);"SHOT";Z
180 INPUTC(Z)
190 INPUTD(Z)
200 PRINTC(Z);D(Z)
230 FORJ=1TO8
240 IFA(J)=C(Z)ANDB(J)=D(Z)THENK=K+1
250 IFA(J)=C(Z)ANDB(J)=D(Z)THENPRINT"HIT SCORE:";K
260 IFK=8THEN430
270 NEXTJ,Z
290 PRINT"■"
300 PRINT
310 PRINT"♠♠TIME IS UP"
320 PRINT
330 PRINT"YOU HIT";K;"ALIENS"
340 PRINT
350 PRINT"MARKSMAN RATING:";INT((K*100)/(1+3*RND(1)))
360 PRINT
370 PRINT"THEY ARE HIDDEN AT:-"
380 PRINT
390 FORJ=1TO10
400 PRINTA(J);B(J)
410 NEXTJ
420 END
430 PRINT"■"
440 FORH=1TO7
450 PRINT
460 NEXTH
470 PRINT"YOU GOT THEM ALL";
```

```
480 FORU=1TO28
490 PRINTCHR$(INT(32*RND(1))+96);
500 NEXTU
510 GOTO470
```

You'll notice a randomly generated character before the word SHOT each time.

To print the character corresponding to a particular number, you just input PRINT CHR$ (number of character). The "marksman rating" at the end adds a little interest, and is actually related to your skill in destroying the aliens. The more you killed(K) the higher your score. It is divided by 1 + 3*RND(1) just to make it a little more interesting.

Colour

Let's have a look now at some of the colour graphics functions. Enter the next program, run it, then examine the listing to work our what it is doing:-

```
10 FORJ=8TO127STEP17
20 POKE36879,J
30 PRINT"■NEXT COLOUR"
40 INPUTA$
50 NEXTJ
```

This shows POKE in use to create a series of different colours.

Location 36879 is one of the VIC's "SYSTEM VARIABLES". It is used to store two pieces of information. (1) The colour of the screen, and (2) The colour of the border. By incrementing our FOR/NEXT loop in steps of seventeen we are changing BOTH the screen-colour AND the border-colour together.

Change line 10 to FOR J = 136 TO 248 STEP 16. Now notice that the screen changes but the border doesn't. Notice that eight new colours have been generated — orange, light-orange, pink, light-cyan, light purple, light green, light-blue and light yellow.

To work out what number you have to POKE into 36879 just LET S = the colour of the screen, and LET B = the colour of the border.

Use the numbers given here:

SCREEN	S	BORDER	B
BLACK	0	BLACK	0
WHITE	1	WHITE	1

RED	2	RED	2
CYAN	3	CYAN	3
PURPLE	4	PURPLE	4
GREEN	5	GREEN	5
BLUE	6	BLUE	6
YELLOW	7	YELLOW	7
ORANGE	8		
LIGHT ORANGE	9		
PINK	10		
LIGHT CYAN	11		
LIGHT PURPLE	12		
LIGHT GREEN	13		
LIGHT BLUE	14		
LIGHT YELLOW	15		

and use the formula 16*S + B + 8 to combine them. Thus to get an orange screen with a white border just use the statement POKE 36879, 16*S + 1 + 8.

Try this:

```
10 INPUT"FIRST COLOUR";S
20 INPUT"SECOND COLOUR";B
30 POKE36879,16*S+B+8
40 GOTO10
```

Run it and input number from the tables.

Now change line 30 to POKE 36879, 16*S + B (ie without the plus eight) and RUN it again, inputting the same numbers. Interesting isn't it?

Not it's time to start POKEing into the VIC's COLOUR memory map area. Don't worry if this sounds complicated — just follow through what's going on.

```
10 FORJ=1TO505
20 PRINT"■";
30 NEXTJ
40 POKE38400+INT(506*RND(1)),INT(8*RND(1))
50 GOTO40
```

Line 10 to 30 just fill the screen up. Line 40 is the interesting one. It POKEs a random number between 1 and 8 (this represents one of eight different colours) into one of the addresses between 38400 and 38906. These are the COLOUR addresses for the screen — one address for each print position. Note that there are 506 such positions, since the screen is 22 x 23.

Now try this one.

```
10 PRINT MID$("■■■■■■■■",INT(6*RND(1))+1,1);"■";
20 GOTO10
```

Line 10 is rather clever — the first part selects a new colour at random. Notice how the function MID$ is used to select a single character from the string of colour controls.

Strings and Ladders

One way of making the most of your memory is to use strings, assigned at the start of the program, for phrases which you intend to use at various parts of the program. The following game SNAKES AND LADDERS shows these techniques in use.

```
5 G$="SCORE IS"
10 A$="SNAKE "
15 H$=":-"
20 B$="LADDER "
25 K$=" WINS"
30 L$="WORTH "
40 PRINT"■";A$;"S AND ";B$;"S"
50 INPUT"PLAYER 1";C$
70 INPUT"PLAYER 2";D$
90 PRINT"TO START GAME"
100 A=0
110 B=0
130 INPUT"PRESS RETURN";E$
150 GOSUB260
155 K=0
160 GOTO280
165 A=A+E
170 PRINTC$;H$;M$;L$;E
180 PRINTG$;A
```

```
190 PRINT
195 IFA>19THEN365
200 K=1
210 GOTO280
215 B=B+E
220 PRINTD$;H$;M$;L$;E
230 PRINTG$;B
232 PRINT
233 PRINT
235 IFB>19THEN390
240 PRINT"FOR NEXT MOVE"
250 GOTO130
260 FORS=1TOINT(3000*RND(1))
265 NEXTS
270 PRINT"□"
275 RETURN
280 C=INT(5*RND(1))+1
290 IFC<3THENT=-1
300 IFC>2THENT=1
310 IFC<3THENM$=A$
320 IFC>2THENM$=B$
330 P=INT(6*RND(1))+1
340 E=P*T
350 IFK=0THEN165
360 GOTO215
365 PRINT"□"
370 PRINTC$;K$,," WITH";A-B;"POINTS"
385 GOTO400
390 PRINT"□"
395 PRINTD$;K$,," WITH";B-A;"POINTS"
400 INPUT"ANOTHER GAME";N$
420 PRINT"□"
430 IFN$="YES"THEN90
440 PRINT"OK.BYE"
450 END
```

This program shows quite clearly how strings can be used to save memory (in fact, the idea was carried a little too far, just to make the technique plain). There are a few other things in the listing from which we can learn. Look at lines 290 to 320. These determine if the player will get a SNAKE (and a negative score) or a LADDER (and a positive score). If line 280 had read C = INT(4*RND(1)) + 1 there would be a pretty good chance that the

game would never end, because each players gains would approximately equal his or her losses, and the players would lose interest long before somebody happened to chance a win. By making the odds in favour of a ladder (a ladder, and positive score are generated about 60% of the time) the program ensures that both players' scores gradually build up.

The next program is based on exactly the same idea as SNAKES AND LADDERS but takes longer to play, and — because it has more variables — is considerably more interesting.

```
1 D$="MILES TO GO"
2 E$="ALL OK"
3 F$="SMASH"
4 G$="COPS"
5 H$="PUNCTURE"
6 J$="POINTS"
7 K$="OIL"
8 L$=" WINS WITH"
9 N$="PETROL"
10 PRINT"□      ROADRACE"
30 INPUT"DRIVER 1";B$
45 A=0
50 INPUT"AND 2";A$
60 B=0
70 INPUT"PRESS RETURN";C$
80 FORJ=1TO299:NEXT
90 PRINT"□";A$
95 K=0
100 GOTO180
105 A=A+Z
110 IFA>390THEN270
115 IFA<0THENA=0
120 PRINTM$;397-A;D$
125 PRINT
130 PRINT,B$
135 K=1
140 GOTO180
145 B=B+Z
150 IFB>390THEN280
155 IFB<0THENB=0
160 PRINTM$;397-B;D$
170 GOTO70
180 C=INT(11*RND(1))+1
```

```
190 IFC>5THENGOSUB210
200 IFC<6THENONCGOSUB220,230,240,250,260
205 IFK=0THEN105
207 IFK=1THEN145
210 M$=E$
212 Z=27
215 RETURN
220 M$=F$
222 Z=-19
225 RETURN
230 M$=G$
232 Z=-7
235 RETURN
240 M$=H$
242 Z=-12
245 RETURN
250 M$=N$
252 Z=-5
255 RETURN
260 M$=K$
262 Z=-23
265 RETURN
270 PRINTA$;L$,39*(A-B)+A-B;J$
275 END
280 PRINTB$;L$,39*(B-A)+B-A;J$
285 END
```

There are a few things to note about this program. It is meant to be a race, and although the drivers in this game actually go backwards, they are unlikely to go backwards beyond their starting point. Therefore, if at anytime the score becomes less than zero, it is reset to zero (lines 115 and 155). Also in this game, as in SNAKES AND LADDERS, there is a slightly better than even chance of getting an ALL OK (and a positive mileage). See if you can find the line in the listing that ensures this.

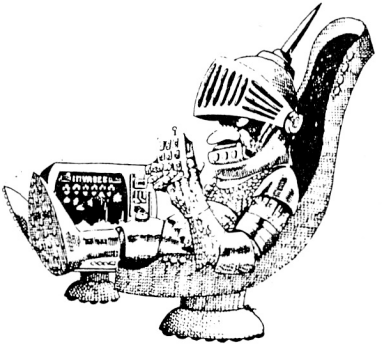
Comparing line 260 in SNAKES AND LADDERS with line 80 in ROADRACE is instructive in the first program, the delay is random (and varies from a delay of practically zero when S = 1 to a longer period when S = 3000). In ROADRACE, the delay is set (purely arbitrarily at 299). There is no reason why you can't set the delay in either program to zero (delete the FOR/NEXT loop, but leave in the CLR) or any number you like, or — if you prefer the unexpected — at a random number. Do not set the random limit too high (like, say, S = 1 TO 10000*RND) because you run the risk of

losing interest in the game if the delay is close to 10000 time and time again.

The next game — 52 BLUFF — uses the string idea again. In this game the computer deals two cards. If you think the next card to be dealt will be between the first two, you place a bet of your choice. This game is more interesting than some computer betting games like FRUIT MACHINE because you can decide on the likelihood of a win and adjust your bet accordingly. You can even decide not to bet at all.

```
10 S=30
20 A$=""
30 F$="CARD1:"
40 G$="CARD2:"
50 H$="CARD3:"
60 A=INT(13*RND(1))+1
70 B=INT(13*RND(1))+1
80 IFB=ATHEN70
90 PRINT"STAKE:£";S
100 PRINTF$;A$;
110 Z=A
120 GOSUB460
130 PRINT
140 PRINTG$;A$;
150 Z=B
160 GOSUB460
170 INPUT"WAGER";C
190 PRINT"□";
200 IFC=0THENPRINT,"COWARD"
210 PRINTA,B
220 D=INT(13*RND(1))+1
230 IFD=AORD=BTHEN220
240 PRINT
250 PRINTH$;A$;
260 Z=D
270 GOSUB460
280 IFA<DANDD<BTHENGOSUB360
290 IFA>DANDD>BTHENGOSUB360
300 IFD>BANDD>ATHENGOSUB410
310 IFD<BANDD<ATHENGOSUB410
320 IFS<1THEN440
330 INPUTK$
```

```
340 PRINT"□"
350 GOTO60
360 S=S+2*C
370 IFS>199THENPRINT"YOU HAVE BROKEN THE   BANK":END
390 IFC<>0THENPRINT"YOU WIN £";2*C
400 RETURN
410 S=S-C
420 IFC<>0THENPRINT"YOU LOSE £";C
430 RETURN
440 PRINT"YOU ARE BROKE"
450 END
460 IFZ<>1ANDZ<=10THENPRINTZ
470 IFZ=1THENPRINT"ACE"
480 IFZ=11THENPRINT"JACK"
490 IFZ=12THENPRINT"QUEEN"
500 IFZ=13THENPRINT"KING"
510 RETURN
```



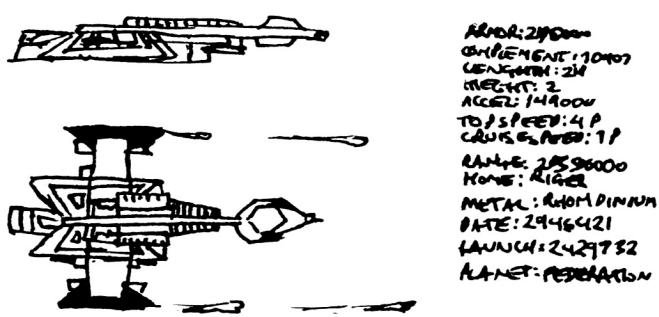
Hot Sauce

You will recall that earlier in this book there was a program in which the computer thought of a number, then gave you hints to help you guess it. You probably realised that if you started with 50 as your first guess, it was pretty easy to narrow down the number by going to either 25 or 75 on the second guess. This method allowed you to get the correct number fairly easily. Here is another program which appears somewhat similar. But it is far harder to work out a system to beat it.

```
10 PRINT"■HOT SAUCE"
30 PRINT
40 PRINT"WHAT'S YOUR NAME,","PARDNER?"
50 INPUTA$
70 PRINT"■"
80 PRINT
90 S=0
100 PRINT"OK ";A$;",I AM"
105 PRINT"THINKING OF A NUMBER"
110 PRINT"BETWEEN 1 AND 100"
120 PRINT"YOU HAVE TWELVE       GUESSES"
130 J=INT(100*RND(1))+1
140 PRINT
150 S=S+1
160 IFS=13THEN420
170 PRINT"WHAT NUMBER AM I      THINKING OF"
175 INPUTA
190 PRINT"■"
200 IFA=JTHEN360
210 IFA<JTHEN300
220 IFA-J<5THENPRINT"BOILING,";A$
230 IFA-J<12ANDA-J>4THENPRINT"HOT"
240 IFA-J<25ANDA-J>11THENPRINT"WARM"
250 IFA-J<45ANDA-J>24THENPRINT"COLD"
260 IFA-J>44THENPRINT"FREEZING,BABY"
270 PRINT
280 PRINT"NEXT GUESS?"
290 GOTO150
300 IFJ-A<5THENPRINT"VERY,VERY CLOSE"
310 IFJ-A<12ANDJ-A>4THENPRINT"PRETTY CLOSE"
320 IFJ-A<25ANDJ-A>11THENPRINT"JUST SO-SO"
330 IFJ-A<45ANDJ-A>24THENPRINT"PRETTY HOPELESS"
```

```
340 IFJ-A>44THENPRINT"PATHETIC,";A$
350 GOTO280
360 PRINT"YOU WERE RIGHT,";A$
370 PRINT"I WAS THINKING OF";J
380 PRINT"SO YOU GET ANOTHER GO"
385 INPUTA$
390 RUN
420 PRINT"□SORRY ";A$;" YOU"
425 PRINT"DIDN'T GUESS IT"
430 PRINT
440 PRINT"I WAS THINKING OF";J
450 END
```

The term "absolute" (ABS on keyboard) was introduced just after the listing for PESKY PIKSY. Refer back if you're not sure what ABS does. The HOT SAUCE program could be written quite differently from the above, by using the ABS in the listing here, the VIC's response to your guess is determined by whether the number you guess is higher or lower than the one it is thinking of, and by the difference between the numbers. If you rewrite HOT SAUCE using ABS, you'll find that only the difference between the numbers will determine the comment ("PATHETIC" or "BOILING" or whatever) the VIC will make. And just as HOT SAUCE is more difficult to play than GUESS MY NUMBER, HOT SAUCE with ABS is harder than without it. As an exercise, write a HOT SAUCE program using ABS. You should find it uses less memory than the above listing.



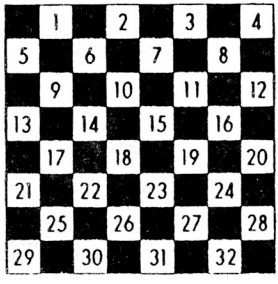
Draughts

This next section will enable you to develop a partially complete game of draughts. However, the main reason it has been included is so I can explain a method of numbering boards for board games which make it easy for a computer to manipulate. A similar board-numbering system can be used as the core of a chess program, noughts and crosses and even Nine Mens Morris. I strongly urge you to follow through the text carefully, entering the program as listed.

If you do this, you should learn quite a lot which will help you when you come to write your own board games.

THE NUMBERING SYSTEM

The normal way to number a draught board is to count off the white squares (actually, the black squares are counted, but white ones are used here because they are easier to use in the computer context) from one to 32. But this numbering method creates a problem when we try to define a move in terms of the difference between two diagonally adjacent squares. In one direction, the difference between the squares can be three or four and in the other direction, the difference can be five or four. There are also no 'spare' numbers to indicate where the edge of the board begins,



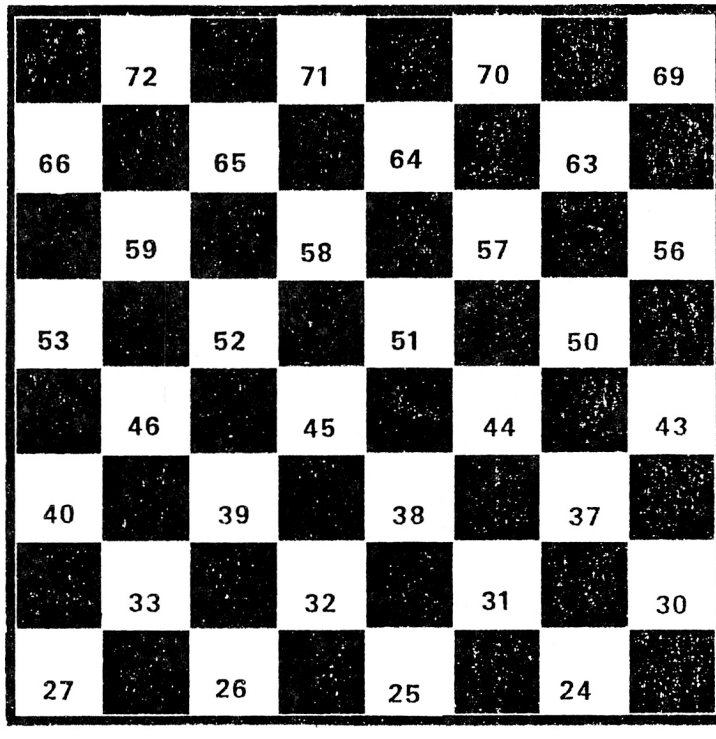
Now, a gentleman by the name of A. L. Samuels wrote an article for Scientific American in the 1960s (see Strachey, Christopher, "Systems Analysis and Programming", in Readings from Scientific American, W H Freeman and Co., San Francisco, 1971) in which he devised a clever numbering system in which the difference between diagonally adjacent squares is always four and five (or minus four and minus five). It also allowed for numbers to be given to squares which were 'off the board'.

I've changed his numbering system a little to make it more convenient for the computer, and in my system, the difference between squares is always six or seven (or minus six and minus seven). My system, very simply, sets up an array of 82, and allots certain elements of the array to squares on the

board. All the others are understood by the computer to be off the board.

In this program, the computer allots the value 9 to any square off the board, zero to an empty square, 1 to a computer's ordinary piece (2 to a computer King) and −1 to the player's ordinary piece (and −2 for a player's King). This may sound a little complicated, but bear with me, and it will (hopefully) all become clear.

Here is my numbered board. You can see that if you move from the top left hand corner (69) to the square diagonally below it (63) the difference between the two spaces is −6. Now, choose any other square on the board from which you can move down and to the left, and you'll see there is a difference of −6 between the square you started on, and the square on which you finished. This sort of predictable result is relatively easy for a computer to handle. Move in the other direction, that is downward and to the right, and you'll see the difference between the two squares is −7. In the first version of draughts, we'll actually be playing on the board printed, so you'd better start looking for a number of small buttons to use as playing pieces.



The program is in two parts. The first 'sets up the board' (the subroutine starting at 9000) and the second (10 to 370) actually plays the game. Your pieces start at the bottom of the printed board (on the lower numbers) and the computer starts at the top. You place the pieces on the board and then press RUN. It plays remarkably fast.

The computer's moves are shown as two numbers. The first is the square it is moving from, and the second — naturally enough — is the square it is moving to. Move its pieces on the board as instructed and then decide on your move. Make sure you move your piece BEFORE you input your move (which you do by entering the number of the square you're moving from, then RETURN, then the square you're moving to) or you may forget what your move was.

Note that there is no provision within the program for multiple jumps by either player or the computer.

The program has been deliberately left incomplete so you can work on it once you've understood the material presented in this section. A number of suggestions for improvement, including the provision of multiple jumps, is included at the end.

Here is the first part of the program, the section which sets up the board. It creates an array (DIM A(82)) then fills the array with numbers representing pieces on the board (1 and −1) empty square (0), and squares off the board (9).

```
9000 DIMA(82)
9020 DIMX(2)
9030 X(1)=-6
9040 X(2)=-7
9050 FORZ=1TO82
9060 A(Z)=9
9070 IFZ<73ANDZ>55ANDNOT(Z=67ORZ=68OR
          Z=60ORZ=61ORZ=62)THENA(Z)=1
9080 IFZ<54ANDZ>42ANDNOT(Z=47ORZ=48OR
                    Z=49)THENA(Z)=0
9090 IFZ<41ANDZ>23ANDNOT(Z=34ORZ=35OR
         Z=36ORZ=28ORZ=29)THENA(Z)=-1
9100 NEXTZ
9110 A$="MY MOVE "
9120 B$="YOURS,"
9130 RETURN
```

Check to see if this is working correctly before you proceed. Add 8000 END, then RUN the program. When it stops, get the computer to print out the following, to make sure it has assigned values correctly. The right answer is given in brackets after the command:

PRINT A(23) *(9)*
PRINT A(54) *(9)*
PRINT A(64) *(1)*
PRINT A(38) *(−1)*
PRINT A(51) *(0)*
PRINT A(73) *(9)*

Now enter the following:

```
1 GOSUB9000
10 Q=0
20 Z=24
25 IFA(Z)=9THEN100
30 IFA(Z)<1THEN100
50 FORX=1TO2
60 IFA(Z+X(X))<0ANDA(Z+2*X(X))=0THENQ=X(X)
80 IFQ<>0ANDZ+2*Q>23THEN120
85 Q=0
90 NEXTX
100 IFZ<72THENZ=Z+1:GOTO25
110 IFQ=0THEN160
120 A(Z+Q)=0
130 A(Z+2*Q)=A(Z)
140 A(Z)=0
150 PRINTA$;Z,Z+2*Q
155 GOTO320
160 Y=0
170 Z=INT(49*RND(1))+23
180 Y=Y+1
190 IFA(Z)<>1ANDA(Z)<>2THEN170
200 FORX=1TO2
210 IFA(Z+X(X))=0THENQ=X(X)
220 IFA(Z)=2ANDA(Z-X(X))=0THENQ=-X(X)
230 IFQ<>0THEN290
240 NEXTX
260 IFY<100THEN170
270 PRINT"YOU WIN"
```

```
280 END
290 A(Z+Q)=A(Z)
300 A(Z)=0
310 PRINTA$;Z,Z+Q
320 PRINT,B$
330 INPUTA,B
340 A(B)=A(A)
350 A(A)=0
360 IFABS(A-B)>7THENA(A+INT((B-A)/2))=0
370 GOTO10
```

Play a complete game with the computer, using little buttons on the board provided, then return to this book and we'll go through the program line by line to explain how it works.

Line 10 sets up Q as the indicator (or 'flag') to show a move has not been made. When a move decision has been made, Q changes to a value which is not zero. Line 20 starts the loop which checks every square on the board to see if a capture can be made. Line 30 checks to see if the square being looked at (ie the element of the array) is a computer piece (1). If it is not, the computer does not waste time checking through the logic statements, and goes to 100.

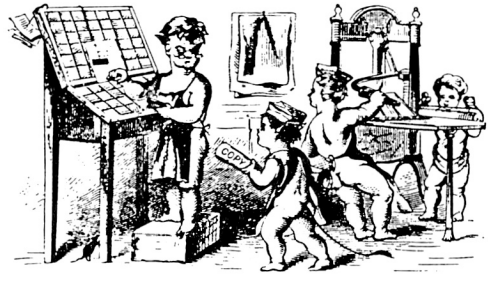
If the computer finds that the piece it is considering **is** one of its own it goes through the X loop, checking that the piece down and to the left (− 6) is less than 0 (if it is, the square contains a player's piece) and that the square twice the distance away (ie − 12) is an empty square. If these two conditions are true it sets the capture flag (Q) to − 6.

If, in line 80, the computer finds Q still equal to zero, it knows it has not made a decision to capture. If, however, Q is non-zero, control is sent to line 120 to act on the decision. Line 90 looks through the X loop with the second value (− 7) of X(X). Line 105 continues to search through the Z loop, checking every square to make sure a potential capture is not missed.

If Q still equals zero when the computer gets to line 110 it knows there is no potential capture on the board, so control is sent to line 160 to find a random, legal move. If, however, Q is non-zero, lines 120, 130 and 140 make the capture, and line 150 displays it. Having made its move, it sends control to line 320 to accept the player's move.

If a legal move has not been made, the computer looks for a random move. The flag Y (line 160) counts the number of attempts made to find a legal move (adding one to Y in line 180 each time it tries a move). If no move is found in 99 attempts (line 260) the computer wisely concedes defeat.

Line 190 checks to see if the square it has generated contains one of its own pieces, and if it does not, and Y is less than 100, goes back to pick



another square at random. Once it finds a piece of its own, it applies the 'X loop test' (lines 200 to 240) to see if it can move legally, again using the flag Q to indicate whether or not a move has been made.

Lines 270 and 280, as we've said, concede defeat.
Lines 290 to 310 make the move and print it for the player.

The player then makes a move (lines 325 and 330) and the computer acts on it (lines 340 to 360) using the somewhat complicated line 360 to capture a piece if the player's move is greater than seven. It knows that a move to an adjacent square must be either 6,7, −6 or −7 and that a move greater than this can only come if the player is jumping over a piece (or cheating). The computer *never* assumes the player is cheating (because the computer cannot cheat) so uses line 360 to work out which square lies between the square the player moved from, and the square the player moved to, 'erase' its own pieces.

Once you've understood the foregoing, you can start improving the program. I'll give you a way of printing up a board (although you'll still have to refer to the numbered board to input your moves) but the other improvements are up to you. Once you've got the printing of the board under control, you might like to try and add the following, to change the program into a real draughts game:

— Add Kings
— Add multiple jumps (these are relatively easy, as each multiple jump is a predictable arithmetic change from the starting square, and the array is large enough to stop you jumping out of it in most cases when looking for potential multiple jumps). Player's multiple jumps are very easy. Just get the computer to ask "IS THIS A MULTIPLE JUMP?" If a capture (line 360) is made, and if the answer is "YES", send control back to 325 to allow the

second jump to be made.

— Add a 'capture tally' so the computer will print up after each move how many each of you have taken. The computer can then declare itself the winner if it, say, captures nine of your pieces.

— Add a mechanism to accept moves directly, rather than by having to look it up on a table.

There are probably other improvements you can make, but I feel these should be your first priority.

The final 'improvement' I will give you to the program is a little routine (developed by Tony Baker) to print out the board.

Delete line 320, change 310 and 150 to GOSUB 5000 add 15 GOSUB 5000, and change 155 to GOTO 330.

```
5000 FORM=24TO72
5010 IFA(M)=1THENA(M)=66
5020 IFA(M)=-1THENA(M)=87
5030 IFA(M)=0THENA(M)=32
5050 NEXTM
5060 PRINT"■";A$;Z;Z+Q
5070 PRINTB$
5080 PRINT
5090 FORK=0TO3
5100 FORJ=0TO3
5110 PRINT"■ ■";CHR$(A(72-J-13*K));
5120 NEXTJ
5130 PRINT
5140 FORJ=0TO3
5150 PRINT"■";CHR$(A(66-J-13*K));"■ ■";
5160 NEXTJ
5170 PRINT
5180 NEXTK
5190 PRINT"■"
5200 FORM=24TO72
5210 IFA(M)=66THENA(M)=1
5220 IFA(M)=87THENA(M)=-1
5230 IFA(M)=32THENA(M)=0
5250 NEXTM
5260 RETURN
```

With this in place, you'll actually see the piece move. Ignore the line "MY MOVE..." when it gives the same square number twice. Wait for it to change, and then watch the computer's piece move. Captures look particularly good. Wait until the cursor appears before you attempt to enter your move.

Note that the two loops (5000 to 5050 and 5200 to 5250) decide which character will represent which piece. By all means change these if you like.

First Steps Towards the Stars

Most computer systems in the world, micro to mainframe, have at least one STAR TREK game in their library. Here is one of the simplest versions of this old favourite. Study of the program will teach you some of the fundamentals of star games, and will give you a core to build on.

```
2 PRINT"□TIMEWARP"
3 PRINT
4 PRINT
5 G=10
6 H=0
7 INPUT"NAME";N$
8 INPUT"TREASURE";T$
11 INPUT"ENEMY 1";E$
13 INPUT"ENEMY 2";F$
16 PRINT"□"
17 J=INT(2*RND(1))
18 IFJ=1THENL$=E$
19 IFJ=0THENL$=F$
20 IFG<1THEN50
21 PRINT"SHIELD THICKNESS:";G
22 INPUTQ$
23 H=H+1
24 PRINT
25 PRINT"TIME LEFT:";17-H
26 PRINT"DANGER-";L$;" AHEAD"
27 INPUTB$
28 GOSUB55
29 IFH=17THEN52
30 K=INT(2*RND(1))
```

```
31 IFK=0THEN35
32 G=G-1
33 PRINT"THE ";L$;" GOT YOU",N$
34 GOTO17
35 G=G+1
36 PRINT"YOU DESTROYED THE",L$;",";N$
37 INPUTZ$
39 PRINT"■YOU ARE CLOSER TO THE ";T$
40 INPUTD$
41 PRINT"■"
42 M=INT(5*RND(1))
43 IFM<4THEN17
44 GOSUB55
45 IFG>0THENPRINT"CONGRATULATIONS,",N$
46 PRINT"YOU GOT THE ";T$
47 PRINT"OUT OF THE TIMEWARP"
48 PRINT"WITH";17-H,"TIME UNITS SPARE"
49 END
50 PRINT"GAME OVER,YOU ARE DEAD"
51 END
52 PRINT"TIMEWARP HAS IMPLODED"
53 PRINT"YOU HAVE FAILED"
54 END
55 FORW=1TOINT(3000*RND(1))
56 NEXTW
57 PRINT"■";
58 RETURN
```

If the value of H could be used to print the "sector of the galaxy" we were in (ie *IF H > 3 AND H < 7 THEN PRINT "YOU ARE IN SIRIUS SECTOR"*) we would be well on the way to creating a real "STAR TREK".

In this next program, for the first time, we access the timer. This allows us to add a time dimension to programs.

```
2 PRINT"■  ■FRENZY"
4 PRINT"PRESS ANY KEY"
5 GETA$:IFA$<>""THEN5
6 GETA$:IFA$=""THEN6
7 G=INT(15*RND(1))+6
8 FORJ=1TOG
9 FORD=1TO50*INT((20+50*RND(1))/J)
10 NEXT
11 PRINT"■"
```

```
12 IFJ>1THENPRINT"TIME LAST GO:-";M
13 PRINT
14 PRINT"TEST NO.";J;"OUT OF";G
15 PRINT
16 PRINT"YOU HAVE";220-7*J;"SECONDS"
17 FORW=0TOINT(11*RND(1))
18 PRINT
19 NEXTW
20 F=(2*G+3*J/2)*J*J-INT(9*RND(1))-1
21 TI$="000000"
23 PRINT"OK DUM DUM,COPY THIS  NUMBER"
24 PRINT
25 X=INT(4*RND(1))
26 IFX=1THENPRINT"      ";
27 IFX=2THENPRINT,
28 IFX=3THENPRINT,"      ";
29 PRINTF
30 INPUTK
31 M=TI
34 IFK=FNDM<220-7*JTHENNEXTJ
35 IFK<>FTHEN50
36 GOTO46
37 PRINT"□"
38 PRINT"YOUR SANITY RATING"
39 PRINT"IS";M*J/8+7*RND(1)
40 PRINT
41 PRINT"COULD YOU STAND IT    AGAIN"
42 INPUTH$
43 IFH$<>"NO"THENGOTO2
44 END
46 PRINT"□YOU ARE FAR TO SLOW"
47 PRINT
48 GOTO38
50 PRINT"□YOU CANNOT EVEN COPY  NUMBERS"
51 PRINT
52 GOTO38
```

There are a few extra ideas in this program which are worth pointing out. Firstly, look at lines 25 to 28. These decide how far across the line the number F will be printed (if X = 0 the number is printed hard against the left because there is no instruction for X = 0). The second point worth

noting is that the manner in which the answer to COULD YOU STAND IT AGAIN? is treated. Any answer at all, except NO. will be interpreted as an instruction to go back for a second game. So if your smart friends input YAY, YEAH, Y, YESSIR, OK or whatever, the VIC will "understand" that YES (actually, NOT NO) is meant.

Music

Let's explore the VIC's music making abilities now. Type in the following lines of program and I'll tell you what they're for.

```
10 S1=36874
20 S2=36875
30 S3=36876
40 S4=36877
50 V=36878
```

Now, S1 stands for Speaker-1, S2 stands for Speaker-2, S3 stands for Speaker-3, and S4 stands for Speaker-4. V is for Volume. The numbers after the equals sign are the VIC's music producing numbers. You must always use these numbers when making music.

To generate SOUND we must use the POKE statement — you may remember we used this earlier on when we were first exploring colour. Let's first see how to generate SILENCE. Add the following lines — these form a subroutine:

```
500 FORA=S1TOV
510 POKEA,0
520 NEXT
530 RETURN
```

Notice that we have POKEd all four Speakers, and the Volume, with zero. Now lets actually start making music. We'll start by exploring the purpose of S1 — Speaker-1. Add the following lines and then RUN the program.

```
100 GOSUB500
110 POKEV,5
120 FORA=128TO255
130 POKES1,A
140 FORB=1TO50:NEXTB
150 NEXTA
160 GOSUB500
170 END
```

Do you see what happens? The number 5 we POKEd into V was the volume level. The number we POKEd into S1 was the tone. You should have discovered this by RUNning it. Change line 130 to POKE S2,A and RUN it again. Now change it to POKE S3,A and RUN it once more. Now for the biggest surprise of all — make line 130 POKE S4,A and RUN it again. S1, S2, and S3 are there to make music — S4 produces noise.

Read and Data

Type NEW. Now input the following program:

```
10 READN
20 PRINTN
30 GOTO10
9000 DATA17,16,42,99,0,57,123
```

Line 10 instructed the computer to READ a new value into the variable N. It got this value (17) from the DATA statement in line 9000. The next time round it READ the value 16 into N, then 42, and so on up to 123. This time when it tried to READ N it had run out of data, and so stopped with the error message OUT OF DATA.

We can use READ and DATA to help the VIC to play a tune. Let's see how this can be done.

```
10 S1=36874
20 S2=36875
30 S3=36876
40 S4=36877
50 V=36878
100 GOSUB500
110 POKEV,4
120 FORA=1TO16
130 READN
140 POKES2,N
150 FORB=1TO1000:NEXTB
160 NEXTA
170 GOSUB500
180 END
500 FORA=S1TOV
510 POKEA,0
520 NEXT
530 RETURN
```

```
9000 DATA135,147,159,135
9010 DATA135,147,159,135
9020 DATA159,163,175,175
9030 DATA159,163,175,175
```

A pretty bad tune eh? I'm sure you can improve it. This table will help. It tells you what values of N you need to POKE in order to make the notes required.

NOTE	*VALUE*	*NOTE*	*VALUE*	*NOTE*	*VALUE*
C	*135*	*C*	*195*	*C*	*225*
C #	*143*	*C #*	*199*	*C #*	*227*
D	*147*	*D*	*201*	*D*	*228*
D #	*151*	*D #*	*203*	*D #*	*229*
E	*159*	*E*	*207*	*E*	*231*
F	*163*	*F*	*209*	*F*	*232*
F #	*167*	*F #*	*212*	*F #*	*233*
G	*175*	*G*	*215*	*G*	*235*
G #	*179*	*G #*	*217*	*G #*	*236*
A	*183*	*A*	*219*	*A*	*237*
A #	*187*	*A #*	*221*	*A #*	*238*
B	*191*	*B*	*223*	*B*	*239*
				C	*240*
				C #	*241*

Maestro

There's no reason why the computer can't be programmed to 'write' its own 'music'. Here are a few programs which do just that. Just change a few of the lines you already have then examine the listings and work out how they do it. Then, write a few similar programs of your own.

DELETE 9000 – 9030
DELETE 130

```
140 POKES1,INT(128*RND(1))+128
```

Then add the following and run the program again:

DELETE 120

```
160 GOTO140
```

You'll have to press STOP to break out of this program, and RUN 500 to stop the music.

Now wipe this, and try the following (which writes its own music, and adds lighting effects).

```
10 S1=36874
20 S2=36875
30 V=36878
40 C=36879
60 DIMN(8)
70 DIMB(4)
80 PRINT"□"
100 GOSUB500
105 POKEV,4
110 FORA=1TO8
120 READN(A)
130 NEXT
140 N=INT(8*RND(1))+1
150 B(1)=N
160 FORA=1TO2
170 GOSUB400
180 B(A)=N
190 NEXTA
200 FORA=1TO4
210 POKES1,N(B(A))
220 FORB=1TO2
```

```
230 GOSUB400
240 POKES2,N(N)
250 POKEC,16*B(A)+N
260 FORJ=1TO250:NEXT
270 NEXTB
280 NEXTA
290 GOTO200
400 N=N+INT(3*RND(1))-1
410 IFN=9THENN=8
420 IFN=0THENN=1
430 RETURN
500 FORA=S1TOV
510 POKEA,0
520 NEXT
530 RETURN
900 DATA195,201,207,209
910 DATA215,219,223,225
```

To break out of the program press STOP and then type GOTO 500. Or if you'd like a simpler program:

```
20 S2=36875
30 V=36878
40 C=36879
60 DIMN(8)
80 PRINT"■"
100 GOSUB500
105 POKEV,4
110 FORA=1TO8
120 READN(A)
130 NEXT
140 N=INT(8*RND(1))+1
240 POKES2,N(N)
250 POKEC,N
260 FORJ=1TO1000:NEXT
290 GOTO140
500 FORA=S2TOV
510 POKEA,0
520 NEXT
530 RETURN
900 DATA195,201,207,209
910 DATA215,219,223,225
```

Again, you need to press STOP and then type GOTO 500 in order to break out.

Doing it in Your Head

The next two programs do not have timers, but could well be adapted to have a real time limitation if you like.

```
10 PRINT"□UNICORNS AND GRIFFINS"
20 G=INT(5*RND(1))+6
30 PRINT
40 PRINT"DEGREE OF DIFFICULTY  (1 TO 5)"
45 INPUTQ
50 IFQ<1ORQ>5THEN40
70 PRINT"PRESS RETURN TO START"
80 INPUTA$
90 FORJ=1TOG
100 FORD=1TOINT(2000*RND(1))
110 NEXTD
120 PRINT"□"
130 FORW=1TO6*RND(1)+1
140 PRINT
150 NEXTW
160 F=2*G+INT(3*J/2)
170 Z=F+INT(10*Q*RND(1)+5*J)
180 PRINT"UNICORN'S NUMBER IS",F
190 PRINT"WHAT DOES GRIFFIN ADD TO"
200 PRINT"MAKE UNICORN'S NUMBER=";Z
210 INPUTK
220 IFK+F=ZTHENNEXTJ
230 PRINT
240 T=F*INT(100*RND(1)+1)+F
250 IFJ=1THENT=0
260 PRINT"SCORE FOR THAT ROUND  WAS";T
270 IFJ<>GANDK+F<>ZTHENGOSUB330
280 IFJ=GTHENGOSUB360
290 PRINT"DO YOU WANT TO TACKLE THE"
300 INPUT"UNICORN AGAIN";H$
320 IFH$<>"NO"THENRUN
325 END
330 IFK+F<>ZTHENPRINT"THE UNICORN BEAT YOU"
340 RETURN
350 PRINT
360 PRINT"YOU HAVE BEATEN THE   UNICORN"
370 PRINT"YOUR IQ IS";T*J+J
380 RETURN
```

This program introduces an idea you can use in many games — the "degree of difficulty". Generally, the "degree" can be used directly to multiply or divide something, or to be added to or taken away from the limits on a FOR/NEXT loop. In other games, you might have to add lines like (if, say A was the degree): IF A = 1 THEN G = 200, or IF A > 7 THEN GOSUB 90. Look back at some of the earlier programs in this book, and work out ways of modifying them in the light of later things you have learned.

The "degree of difficulty" can easily be worked into the following program. However, as it becomes more and more difficult already as it proceeds, it might be better to make it a little easier before adding the option of increasing the difficulty. Lines 170 and 280 are the ones to modify to make the game simpler, and it is here that the "degree" factor can be added:

```
10 PRINT"□ECHO CHAMBER"
20 Z=INT(8*RND(1))+1
30 FORG=1TOZ+5
40 D=INT(4*RND(1))+1
50 IFD=1THENA$="KIDDO"
60 IFD=2THENA$="SMART ONE"
70 IFD=3THENA$="GENIUS"
80 IFD=4THENA$="COMPUTER FREAK"
90 K=INT(100000*RND(1))
100 PRINT
110 PRINT
120 PRINT"TRY NO.";G;"OUT OF";Z+5
130 PRINT
140 PRINT"THE NUMBER YOU HAVE TO"
150 PRINT"REMEMBER,",,A$
155 FORI=1TO3000:NEXT
160 PRINT"IS";K
170 FORJ=1TO800/G
180 NEXTJ
220 GOSUB280
230 PRINT"OK ";A$;",WHAT WAS"
240 INPUT"THE NUMBER";H
260 IFH=KTHEN330
270 GOTO420
280 FORJ=1TO500*G
290 PRINT"□"
300 NEXTJ
310 RETURN
```

```
330 PRINT"■YAYA,";A$,"YOU GOT IT RIGHT"
340 PRINT"WHEN READY FOR THE    NEXT ONE,"
350 INPUT"PRESS RETURN";C$
370 IFC$<>""THENSTOP
380 PRINT"■"
390 IFG=Z+5THEN440
400 NEXTG
420 PRINT"■YOU BLEW IT,";A$
430 END
440 PRINT"YOU SURE HAVE A","PRODIGIOUS"
450 PRINT"MEMORY,";A$
460 END
```

In this program, which puts a number in the range from zero to 99999 on the screen, and then asks you to remember it for a time period which gets longer with each new number, note that lines 40 to 80 control what A$ will be in each run through the master FOR/NEXT loop. A similar approach was used in SNAKES AND LADDERS and ROADRACE. Line 280, as you can easily see, determines how long the delay loop will be. Have a look at the listing for line 330, noting the comma between the words IT and RIGHT. If the comma was not here, the word RIGHT would scroll around, half on one line and half on the other, when D equals 4. Can you see why?

More things to do with your VIC

Lets look at some other things you can do with your VIC.

Many ideas which are put forward for personal computers — such as running the heating, or for use as a diary or telephone directory — are really not practical. These are the kind of things which can be done far more easily and efficiently with a set of cards and a pen, or a piece of paper and a pen, than they can with the computer. For the time being, given the memory available on your VIC, the need to load programs into it byte by byte from an outside source, and the fact that it needs to be plugged in all the time (and run a display screen as well) to be of any use at all, precludes it from those sorts of applications.

Many people find they initially use their VICs, regardless of why they think they bought them, to play and write games. From this they learn familiarity with the computer, and learn to program. But, many people ask, what then?

This section of the book will look at some of the possibilities.

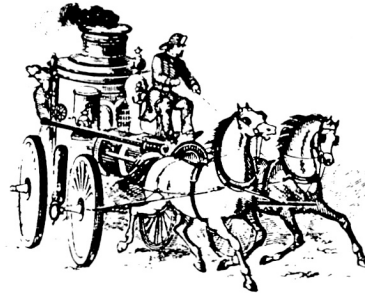
RECORD KEEPING: The VIC is very useful as an adjunct to keeping records like names of members of your club, or the records or cassettes in your music collection, or the numbers of trains you have spotted IF, and this is a big if, you want to sort the information into order (such as alphabetical, or highest to lowest, or by some other category such as age), or you wish to extract something out about it (such as the number of club members you have who live in Cambridge, or the number of records you have by The Beatles which were released before 1970). Unless you want to do this sort of manipulation with the information you have, the data is best kept on cards, but it is an invaluable service if you need to work with the data in some way.

INVENTORY MANAGEMENT: The size of the memory you have will obviously dictate the kind of inventory records you can keep. If you, or your parents, run a small store, you may well find that you can divide the stock into types, and keep one type (such as canned goods) in one cassette data file, Alternatively, if you run a pub, you may find the kinds of goods you have to keep in stock (15 types of beer, snacks like crisps and peanuts, various mixers and whatever) can be fairly easily kept within the memory you now have. You can fairly easily write a program into which you enter the sales from each day, and at the end of each week, the program tells you how much to order.

STAFF PAYROLL: If you have a small firm, you may well find that the time taken to write a program to work out what you should pay each employee each week will be well rewarded. Such information as tax could be automatically calculated, plus other deductions, and if you hook up a printer to your VIC, there is no reason why you can't get the computer to print out the pay slip for you. Of course, unless you have employees with sufficiently different pay levels and deductions to make this worthwhile, you may prefer to continue with your present pencil, paper and calculator method of working out pay.

ROUTINE CORRESPONDENCE: Without a printer, you cannot do this. But you may find, especially if you are organiser of a club which requires you to correspond with a reasonably large membership, that a standard letter — with provision for adding the person's name and address at the beginning, plus some personalised material at the end of the letter — could be used to print out the letters as you demand.

EDUCATION: This is a very big area, where your VIC can be used with good effect. Not only for the kind of material we've discussed in the education section of this book, but for more sophisticated question/answer/grading programs that will teach you as much when you're entering the program as when you're actually running it. No matter which subject you're studying, you'll probably be able to discover an enjoyable way to incorporate the VIC into your study, and — as I said — you'll probably learn as much from programming it to help you as you will from running the program.



It is very simple to create a number of simple interactive programs to use with young children, which will help them recognise such things as counting sequences, letters of the alphabet, simple arithmetic. If you use it with young children, never force them to stay at the VIC for a moment longer than they want to. They'll think of it as a game which they'll return to over and over again if they are not forced to do so, and the familiarity they'll gain with working with computers will be worth as much, in terms of their education, as will the material they'll learn from the program.

USERS CLUBS: You're sure to find other people in your area with VICs. A simple ad: "VIC addict seeks others with similar affliction in Preston, phone XCB HHDG" could bring other owners crawling out the woodwork, all anxious to talk about their computers, and to share programs and ideas. All over the country, small groups of five to a dozen people meet in each others homes to swap ideas, programs, hardware tips, and sort out problems. If you find your parents or partner not entirely literate in the computer field, you may well become 'starved' for someone to talk to. Starting your own local users' group, rather than waiting for someone else in the area to start one, is a great way to meet other owners in the area.

PERSONAL ACCOUNTS: This is another area where your VIC can be very useful, especially if you do not use it as the final source of information about the state of your cheque book or whatever, but rather decide to use the computer program as a general indication of the mess (or otherwise) your personal finances are in. Final totting up should be done with the old standby of calculator, pencil and paper.

SIMULATING: This is an area which can make great demands on your programming ability, but which can provide many enjoyable and useful hours at the computer. If, for example, you wanted to find out what you should do with your stocks and shares, you could feed them in, with their current market values, along with the way the prices for these have

changed in the past month, and project the trend forward for a month. This could indicate not only which stocks you should sell, or which you should get rid of, but just what value your portfolio is likely to have at the end of that period.

Of course, this is a simplistic example, and one which would hardly give very useful information on what you should do with that bulging portfolio, but it may give you some idea.

Another idea: You could feed in your annual salary over the past, say, five years, noting how it has changed from year to year, add a counter-indicator on the rates of inflation in those years, and find out not only how your real spending power has changed in the period, but assuming the trends you've observed continue, how it will grow (or, horrors, shrink) in the coming years.

MAKING MONEY: Ah, you say, now we come to something really worthwhile.

I will start with a warning. Many of these ideas will seem impracticable, and will not be applicable by you, but they may well start you thinking of the kinds of things you could do with your VIC which could enhance your income a little (or even a lot).

ARTICLES: Many people have found that writing articles for computer magazines is a good way to enhance their income. To write an article which is accepted, and the magazines in Britain tend to have far, far more articles than they can ever print, you must work out three things:

(1) Exactly what one thing will the article teach/demonstrate/discuss?
(2) What sort of computer owners (ie what machine, or price bracket, or user, such as hobbyist, businessperson or whatever) will be likely to be interested in the article?
(3) Which magazine will it be sent to?

Unless you can answer those three questions, you haven't got an article, you've got a vague idea or a concept. And no-one will publish that. Buy the computer magazines, all of them if you can afford to, but certainly buy the ones that you intend to write for. In mid to late 1981, the British magazines were all paying between £17 and £45 per article, with more for photographs, programs and diagrams if these were related to the article.

WHERE DO YOU SEND THEM? The major magazines in this country are:

PRACTICAL COMPUTING, IPC, Quadrant House, The Quadrant, Sutton, Surrey, SM2 5AS (01-661 3500)
YOUR COMPUTER — at the same address as PRACTICAL COMPUTING
PERSONAL COMPUTER WORLD, Sportscene Publishers, 14 Rathbone Place, London, W1P 1DE (01 - 637 7991/2/3)

COMPUTING TODAY, 145 Charing Cross Road, London, WC2H OEE. (OI -437-1002/7)
MICROCOMPUTER PRINTOUT, — and VIC Computing, PO Box 48, Newbury, RB16 OBD (0635 201 131)
The main contacts at these publications are (in September, 1981):

PRACTICAL COMPUTING — Peter Laurie
YOUR COMPUTER — Duncan Scot
PERSONAL COMPUTER WORLD — David Tebbutt
COMPUTING TODAY — Henry Budgett
MICROCOMPUTER PRINTOUT and VIC Computing — Richard Pawson

You'll find there are a number of small publications (such as VIC Computing) which are 'machine-specific'. They are often printed by users' groups or by the manufacturer, but welcome, and may pay for, suitable articles.

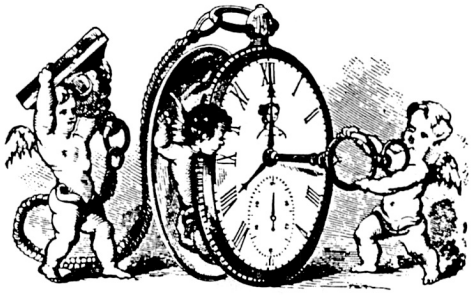
There are also many other computer magazines in Britain (such as EDUCATIONAL COMPUTING (Robin Bradbeer is the contact there) and WHICH Computer) which deal with more specific markets. If you can write for these markets, pick up a copy of the magazine from your local shop, and see what sort of style the articles are in, and make sure yours fits the pattern.

BOOKS: Program collections which are aimed at a specific machine, or machines, seem to sell best. Wander into your local computer shop, if you have one, or into one of the bigger W H Smiths, or Foyles in Charing Cross Road, London, and have a look at the kind of books they are selling. Lion House, in Tottenham Court Road, London (leave Tottenham Court Road tube station by the exist to the Dominion Theatre, then walk **away** from Centrepoint and you'll come to it) have a very wide range of computer books which will give you more ideas.

You can write a book of programs, or a book on how to write programs, or — if you're a hardware buff — how to build peripherals for computers. Booklets, even leaflets, that tell owners how to convert listings given for other machines into their own language can be very popular, especially if you add a few extra programs in as well.

You could write a book to help people choose the right computer for their needs, explaining the features and drawbacks of each one. An "idiot's guide" explaining how computers work could be popular (if you do write such a book, please send me a copy, I'd love to find out), or a guide to lead a first-timer through the jungle of computer jargon.

There seems to be an infinite market for books on how to program in BASIC. No matter how many books are published, there seems to be room for just 10 more.



Have a look at other computer books. You'll find many of these will enhance and expand your interest in your computer. Some books I've found useful are:

Game Playing in BASIC — Donald D Spenver (Hayden)
The First Book of Microcomputers — Robert Spencer (Hayden)
Z80 Software, Gourmet Guide and Cookbook — Nat Wadsworth (Scelbi)
Fifty BASIC Exercises — J P Lamoitier (Sybex)
Computer Programming in BASIC for Everyone — T Dwyer and M Kaufman (Radio Shack)
Microsoft BASIC — Ken Knecht (dilithium Press)
The BASIC Cookbook — Ken Tracton (Radio Shack)
BASIC Computer Games — edited David Ahl (Creative Computing Press)
More BASIC Computer Games — edited David Ahl (Creative Computing Press)
Computer Programs That Work! Lee, Beech and Lee (Sigma Technical Press)
Computer Games for businesses, schools and homes — J V Nahigian and W S Hodges (Winthrop) Comp
57 Practical Programs and Games in BASIC — Ken Tracton (TAB Books)

You may find some material of interest in books published by Database Consultancy, which include:
GETTING ACQUAINTED WITH YOUR ACORN ATOM — Trevor Sharples & Tim Hartnell
GETTING ACQUAINTED WITH YOUR ZX81 — Tim Hartnell
MASTERING MACHINE CODE ON YOUR ZX81 OR ZX80 — Tony Baker
50 RIP-ROARING GAMES FOR THE ZX80 AND ZX81 — edited Jeff Weinrich

If you've written a computer book which you'd like us to publish, send us a letter with some information about the book, and we'll let you know if your idea is of interest, Just write to: INTERFACE, Book publishing, 44 - 46 Earls Court Road, London, W8 6EJ.

Cave Master

You will recall that, earlier in this book, a program under the imaginative name of TIME WARP was listed. The next program uses the same basic program to produce something a little more down to earth. All of the programs in this book can (and should) be developed by you in whatever direction you prefer. Only by doing this will you develop your own programming skills. Anyway, here is one way TIME WARP can be warped.

```
1 G=9
2 PRINT"□CAVE MASTER"
3 GOSUB57
4 H=0
5 E$="CRAZED WIZARD"
6 F$="WICKED WITCH"
9 J=INT(2*RND(1))
10 IFJ=1THENL$=E$
11 IFJ=0THENL$=F$
12 IFG<1THEN47
13 GOSUB57
14 PRINTCHR$(192-G);" AURA TONE";G
15 H=H+1
16 IFH=17THEN50
17 GOSUB57
19 PRINT"LEVEL OF MAGIC";17-H
20 PRINT"HORRORS,";L$,"AHEAD"
21 INPUTB$
22 GOSUB53
23 IFH=17THEN50
24 K=INT(2*RND(1))
25 IFK=0THEN31
26 G=G-2
27 GOSUB57
28 PRINT"THE ";L$,"ZONKED YOU"
29 INPUTA$
30 GOTO9
31 G=G+1
32 PRINT"YOU ZAPPED THE",L$
33 INPUTD$
34 GOSUB53
35 GOSUB57
36 PRINT"THE FOOLS GOLD IS"
```

```
37 PRINT"WITHIN YOUR GRASP"
39 INPUTG$
40 M=INT(6*RND(1))+1
41 IFM<6THEN9
42 GOSUB53
43 GOSUB57
44 IFG>0THENPRINT"YOU DID IT"
45 PRINT"WITH";17-H,,"MAGIC SPELLS LEFT"
46 END
47 GOSUB57
48 PRINT"DEATH COMES TO US ALL"
49 END
50 GOSUB57
51 PRINT"YOU TOOK TOO LONG"
52 GOTO48
53 FORW=1TOINT(1000*RND(1))
54 NEXT
55 PRINT"■"
56 RETURN
57 PRINT
58 PRINT
59 PRINT
60 PRINT
61 RETURN
```

Alpha

In this game, the VIC thinks of a letter of the alphabet, and you have to guess it. Demanding, huh?

```
10 D=RND(-TI)
29 D=0
30 C=1
40 A=INT(26*RND(1))+65
50 PRINT"I AM THINKING","OF A LETTER"
60 PRINT
70 PRINT"ENTER YOUR GUESS";C
```

```
80 GETA$
90 IFA$=""THEN80
110 IFASC(A$)=ATHEN190
120 PRINT"■";A$
130 PRINT"TRY CLOSER TO THE"
140 IFASC(A$)<ATHENPRINT"END";
150 IFASC(A$)>ATHENPRINT"START";
160 PRINT" OF THE ALPHABET"
170 C=C+1
180 GOTO60
190 PRINT"■YES,I WAS THINKING OF ";A$
200 PRINT
210 PRINT"THAT TOOK";C"GUESSES"
220 PRINT
230 IFC<DORD=0THEND=C
240 PRINT"YOUR BEST SCORE THIS  GAME IS";D
250 INPUT"PRESS RETURN";A$
260 PRINT
270 GOTO30
```

Turning the Tables

There are many, many programs in which the computer thinks of a number, and the human player has to guess it. There are two of them in this book. This next program, written by Trevor Sharples, turns the tables.

```
2 PRINT"MYSTIC"
4 PRINT
6 PRINT"THINK OF A NUMBER","BETWEEN"
8 PRINT"ONE AND 100 AND I","WILL GUESS IT"
9 PRINT
10 PRINT"PRESS RETURN TO PLAY"
12 INPUTA$
14 GOSUB86
16 T=0
18 Z=100
20 Y=1
22 X=INT(100*RND(1))+1
24 PRINT"I GUESS";X
26 PRINT
28 PRINT
30 PRINT,"RIGHT(R) ORWRONG(W)"
32 T=T+1
34 INPUTB$
36 IFB$="R"THEN66
38 GOSUB86
40 PRINT"MY GUESS WAS";X
42 PRINT:PRINT"HIGHER(H) OR"
44 INPUT"LOWER(L)";C$
48 GOSUB86
50 IFC$="L"THEN62
52 Y=X
54 S=Z-Y
56 X=Y+INT(S*RND(1))+1
58 IFX=S+YTHEN56
60 GOTO24
62 Z=X
64 GOTO54
66 GOSUB86
68 PRINT"BOY,AIN'T I DE SMART  ONE?"
70 PRINT"I GOT IT IN JUST";T
72 PRINT"TRIES":PRINT:PRINT"COULD YOU FACE ANOTHER"
74 INPUT"GAME,BUD";D$
78 GOSUB86
80 IFD$="YES"THEN4
82 PRINT"BYE BYE THEN WET","BLANKET"
84 GOTO82
```

```
86 FORF=1TOINT(RND(1)*5000*RND(1))
88 NEXT
90 PRINT"□";
92 RETURN
```

Nine Lives

In HANGCAT one player inputs a word, one letter at a time, pressing RETURN between each letter. Then the second player tries to guess the word, pressing RETURN between each letter he or she wants to try. If the second player is wrong, he or she loses a life (hence the title). If the guess is right, the program prints out the correct letter in its correct position in the word.

```
1 PRINT,"HANGCAT"
2 PRINT"PLAYER 1,TYPE IN A    WORD OF 6 LETTERS OR  LESS"
3 PRINT
4 PRINT
5 G$="-"
6 H$="-"
7 J$="-"
8 K$="-"
9 L$="-"
10 M$="-"
11 T=0
12 INPUTA$
13 INPUTB$
14 INPUTC$
15 INPUTD$
16 INPUTE$
17 INPUTF$
19 PRINT"□READY TO PLAY?YOU","HAVE 9 LIVES"
20 INPUT"WHAT IS YOUR GUESS";X$
22 PRINT"□"
23 IFX$=A$THENGOSUB30
24 IFX$=B$THENGOSUB33
25 IFX$=C$THENGOSUB36
26 IFX$=D$THENGOSUB39
27 IFX$=E$THENGOSUB42
28 IFX$=F$THENGOSUB45
29 GOTO48
```

```
30 G$=A$
31 T=T-1
32 RETURN
33 H$=B$
34 T=T-1
35 RETURN
36 J$=C$
37 T=T-1
38 RETURN
39 K$=D$
40 T=T-1
41 RETURN
42 L$=E$
43 T=T-1
44 RETURN
45 M$=F$
46 T=T-1
47 RETURN
48 PRINT
49 PRINTG$;H$;J$;K$;L$;M$
50 PRINT
51 IFG$=A$ANDH$=B$ANDJ$=C$ANDK$=D$ANDL$=E$ANDM$=F$THEN68
52 PRINT
53 T=T+1
54 IFT=9THEN59
55 PRINT"YOU HAVE";9-T,"LIVES LEFT"
56 PRINT
57 PRINT
58 GOTO20
59 PRINT"YOU'RE DEAD"
60 PRINT"THE WORD WAS",A$;B$;C$;D$;E$;F$
61 INPUT"ANOTHER CAT";U$
63 PRINT"□"
64 IFU$<>"NO"THEN2
65 END
66 PRINT
67 PRINT
68 PRINT"YAY CAT"
69 GOTO60
```

Nim

The random number generator cooks up an ovenful of buns, which you and the computer proceed to scoff. There is a limit to how many buns you can eat at a time, and the loser is the person who eats the last bun. It is fairly easy to write a program which will never lose this game, but such a program rapidly becomes boring, if not infuriating. Therefore, this program has built-in fallibility, although it will still win more than half the games it plays.

```
2 PRINT"□";TAB(8);"BUN FIGHT"
3 M=0
4 L=0
5 Z=INT(20*RND(1)+11)
6 H=INT(3*RND(1)+4)
7 IFM=0THENPRINT"SCONES LEFT";Z,"THE MOST YOU CAN TAKE IS";H
8 IFE>0ANDM=0THENPRINT"YOU TOOK";E,"I TOOK";Q
9 FORK=1TOZ
10 PRINT"● ";
11 IFRND(1)<.35THENPRINT
12 NEXTK
13 PRINT
14 PRINT
15 IFM=1THENPRINTTAB(8);"YOU WIN"
16 IFM=2THENPRINTTAB(12);"I WIN"
17 IFM>0THEN15
18 PRINT
19 PRINT"HOW MANY WILL YOU":INPUT"TAKE";E
21 IFE<1ORE>HTHEN19
22 Z=Z-E
23 IFZ>0THEN26
24 M=2
25 GOTO34
26 Q=Z-1-INT((Z-1)/(H+1))*(H+1)
27 IFQ=0THENQ=INT(H*RND(1))+1
28 IFQ>ZTHEN26
29 IFZ>F+2ANDRND(1)<.5THENQ=Q+INT(RND(1)-.5)
30 IFQ>HORQ<1THEN26
32 Z=Z-Q
```

```
33 IFZ=0THENM=1
34 PRINT"□"
35 GOTO7
```

Life

```
1 PRINT"□"
10 K=0*RND(-TI)
20 DIMA(63),B(63),E(7)
30 FORA=0TO7
40 READE(A)
50 NEXTA
60 FORA=0TO63
70 B(A)=32
80 NEXTA
90 FORA=1TO13
100 M=INT(64*RND(1))
110 B(M)=79
120 NEXTA
130 GOTO250
140 FORA=0TO63
150 H=0
160 FORE=0TO7
170 J=A+E(E)
180 IFJ<00RJ>63THEN200
190 IFA(J)=79THENH=H+1
200 NEXTE
210 IFH<>2THENB(A)=32
220 IFH=3THENB(A)=79
230 NEXTA
240 K=K+1
250 PRINT"■GENERATION";K
260 PRINT
270 FORA=0TO63
280 A(A)=B(A)
290 NEXTA
300 FORA=0TO7
310 PRINT
320 FORB=0TO7
```

```
330 PRINT"■";CHR$(A(63-A-8*B));
340 NEXTB,A
350 PRINT"□"
360 FORA=0TO7
370 PRINTTAB(9);
380 FORB=0TO7
390 PRINTCHR$(A(A+8*B));
395 NEXTB
400 PRINT
405 NEXTA
410 INPUTA$
420 IFA$=""THENEND
430 GOTO140
440 DATA9,8,7,1,-1,-7,-8,-9
```

Cubik

This program produces a two-dimensional version of that cube puzzle which recently swept the world. In CUBIK, you have a flat surface divided into 16 segments, numbered (of course) from one to 16. When you run the program, you'll see a flat surface, with the numbers one to four arranged neatly on it. This is the arrangement you want to end up with after rotating the pieces in the CUBIK. The program randomly twists the CUBIK, and you have to get it back in order. You do so by entering the number of the square you wish to 'rotate'. It moves this square, and the other three around it, one square clockwise. Run it, and you'll understand. The key to the square number to enter for rotation is:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

```
10 F=RND(-TI)
20 F=1
30 DIMA(4),B(16)
40 B=1
50 FORD=1TO4
```

This routine sets up the initial CUBIK arrangement.

```
60 C=0
70 IFB=1THENC=1
80 IFB=3THENC=2
90 IFB=9THENC=3
100 IFB=11THENC=4
110 B(B)=C
```

```
120 B(B+1)=C
130 B(B+4)=C
140 B(B+5)=C
150 C=0
160 IFB=1THENC=2
170 IFB=3THENC=6
180 IFB=9THENC=2
190 B=B+C
200 NEXTD
210 GOSUB370
220 IFF<11THENX=INT(11*RND(1))+2
230 IFF>10THENINPUTX
240 IFX=5ORX=9THEN220
```

The next routine makes the twist.

```
250 A(1)=B(X)
260 A(2)=B(X+4)
270 A(3)=B(X+3)
280 A(4)=B(X-1)
290 B(X)=A(4)
300 B(X+4)=A(1)
310 B(X+3)=A(2)
320 B(X-1)=A(3)
330 GOSUB370
340 IFF<10THENINPUTU$
350 F=F+1
360 GOTO220
```

This routine prints the CUBIK.

```
370 PRINT"□"
380 PRINT
390 FORB=1TO16
400 PRINTB(B);
410 IFB/4=INT(B/4)THENPRINT:PRINT:PRINT
420 NEXTB
430 IFF<10THENPRINT"TWIST";F
440 IFF>10THENPRINT"MOVES SO FAR";F-10
450 RETURN
```

Outsmart

A very brief program in which you have ten guesses to find the number (between 1 and 200) chosen by the VIC. The clue given in line 90 takes a bit of getting used to, but once you've played three games, you should be able to 'read' it without any difficulty.

```
20 A=INT(200*RND(1))+1
30 PRINT"□"
40 FORB=1TO10
50 PRINT"GUESS NO.";B
60 INPUTC
80 IFC=ATHEN140
90 PRINT"WRONG";C,,"CLUE-"SQR(ABS(A-C))
100 NEXTB
120 END
140 PRINT"YES,YOU GOT IT IN";B
```

Logica

This is a game in which you try to find a missing RAM chip on a 15 x 15 grid, using nothing but your wits and a patented RAM-detector.

```
10 PRINT"□"
20 A=INT(15*RND(1))+1
30 B=INT(15*RND(1))+1
40 C=1
60 PRINT"RAM IS HIDING","SEARCH";C
70 INPUTD
80 INPUTE
90 IFA=DANDB=ETHEN170
110 PRINTD;",";E;"NOT THERE"
130 PRINT"RAM DETECTOR READS:  ";(A-D+1)*(B-E+1)/10
140 C=C+1
150 GOTO60
170 PRINT"YOU FOUND IT"
180 PRINT"YOU SCORED";300-9*C
185 FORI=1TO500
186 NEXT
190 GOTO170
```

In this program, the RAM is hidden at A and B, with C counting the number of guesses you make. Line 130, the 'RAM detector' gives a very

useful read-out, and within two or three games you should have mastered its output. Note that unlike some 'hunting on a grid' games there is no limit to the number of guesses you can make. However, you should obviously try and find the RAM in as short a time as possible. Your score (line 180) is directly related to the number of attempts you had to make to find the thing.

Blackjack

John Scarne, in his authoritative SCARNE'S ENCYCLOPEDIA OF GAMES, says Blackjack is "the most widely played banking card game in the world". It is relatively simple to play: The players try to get as close as possible to a total of 21, without exceeding 21. Aces count as either 1 or 11, and Kings, Queens, and Jacks each count as 10. This program automatically assigns a value of 1 to an ACE if counting it as 11 would force the total over 21. The human player always goes first in this version of the game. After each card is handed out, you have the option of taking another one, or "standing", that is staying as you are (see line 80). The round is a draw if both of you reach the same total, and it is less than 21. If you "bust", that is you exceed 21, the VIC wins that round automatically. The PRINT lines in this program are a good example of computer arrogance.

```
1 PRINT"■"
10 GOTO160
20 CA=INT(11*RND(1))+1
30 IFCA=11ANDD+CA>21THENCA=1
35 D=D+CA
40 RETURN
50 CA=INT(11*RND(1))+1
60 IFCA=11ANDB+CA>21THENCA=1
65 B=B+CA
70 RETURN
80 PRINT"ANOTHER CARD(1)OR WILL  YOU STAND(0)"
85 INPUTG
100 RETURN
110 PRINT"ANOTHER GAME,","CARD SHARP"
120 INPUTA$
130 PRINT"■"
140 IFA$<>"YES"THENEND
150 GOTO160
160 D=0
```

```
170 B=0
180 GOSUB20
190 H=CA
200 GOSUB20
210 A=CA
220 GOSUB50
230 E=CA
240 GOSUB50
250 F=CA
260 B$="THE COMPUTER HAS"
270 C$="THE HUMAN HAS"
280 PRINTB$;H
290 PRINTC$;E;"AND";F
300 PRINT"TOTALLING";E+F
310 D=H+A
320 B=E+F
330 IFB=21THEN440
340 GOSUB80
350 IFG=1THEN490
355 INPUTZ$
360 PRINT"□"
365 IFD<17THEN530
370 IFD<>21THENPRINTB$;D
380 IFB<>21THENPRINTC$;B
390 IFB=DANDB<>21THENPRINT"SO THIS ROUND'S A DRAW"
400 IFD=21ANDB<>21THENPRINTB$," BLACK JACK..."
405 IFB>21THENPRINTC$;" BUSTED","SO VIC WINS..."
410 IFD<=BORD>21THEN415
412 PRINT"VIC DESTROYS HUMAN    WITH A BRILLIANT",
      "DISPLAY OF CARD","PLAYING"
415 IFD>21THENPRINTB$;" BUSTED"
420 IFB>DANDB<=21THENPRINT"YOU WON
      SOMEHOW...","LUCK I GUESS"
430 GOTO110
440 PRINTC$;" BLACKJACK"
460 IFD<>21THEN370
470 PRINT"BUT SO HAS THE CLEVER COMPUTER
        SO IT'S A    DRAW"
480 GOTO110
490 GOSUB50
500 PRINTC$;" CARD";CA,"TOTAL:";B
510 IFB>21THEN405
515 INPUTZ$
520 GOTO340
```

```
530 PRINTB$;D
535 INPUTU$
540 GOSUB20
560 PRINTB$,"CARD";CA
570 PRINT"SO ITS TOTAL IS";D
575 INPUTU$
580 IFD>21THEN415
590 IFD<17THEN540
600 GOTO370
```

Kalki, The Mind Reader

This game is self-explanatory. Just input it, RUN, and follow the directions. Once you've played it a few times, you can change the PRINT statements to create a totally new game.

```
10 PRINT"□"
20 GOTO90
30 INPUTB$
40 PRINT"□"
50 FORJ=1TO5
60 PRINT
70 NEXTJ
75 PRINT"KALKI, THE MIND-READERSAYS"
80 RETURN
90 GOSUB50
100 PRINT"WHO OWNS THIS COMPUTER"
110 INPUTC$
130 PRINT"PRESS RETURN AFTER","EACH STEP"
140 GOSUB30
150 PRINT"THINK OF A NUMBER"
160 GOSUB30
170 PRINT"DOUBLE YOUR NUMBER,","ADD 4"
180 GOSUB30
190 PRINT"DIVIDE BY 2,","THEN ADD 6"
200 GOSUB30
210 PRINT"SUBTRACT THE NUMBER,","YOU FIRST THOUGHT OF"
220 GOSUB30
230 PRINT"SUBTRACT 3, ","THEN MULTIPLY BY 5"
240 GOSUB30
```

```
250 PRINT"SUBTRACT 3 AGAIN AND  DIVIDE BY 2"
260 GOSUB30
270 PRINT"WRITE DOWN THE NUMBER YOU'VE GOT"
280 GOSUB30
290 PRINT"NOW INPUT THE MONEY   IN YOUR POCKET"
300 INPUTA
310 GOSUB40
320 PRINT"THE NUMBER WRITTEN    DOWN IS 11"
330 GOSUB50
340 PRINT"WHEN YOU LEAVE,PLEASE"
350 PRINT"GIVE ";C$,"£";(A+5)/5,"AS A DONATION"
360 END
```

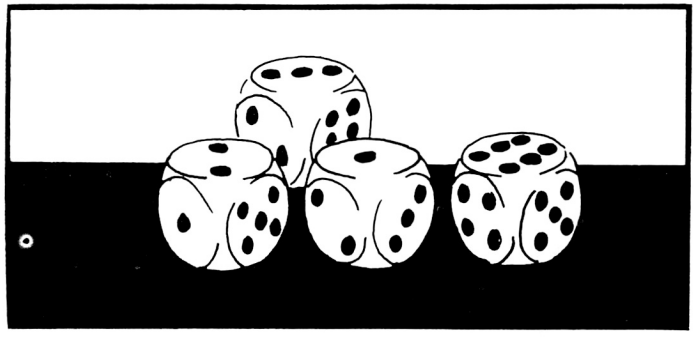
Chemin de Fer

Baccarat was first introduced into France from Italy in about 1490, during the reign of Charles VIII. It is most unlikely, historians say, that Charles played it on a VIC. The Italian game was called Baccara, and this game — Chemin De Fer — is a distant cousin of that old favourite. VIC Chemin De Fer is based on a dice version of the casino game which is usually played with cards. You and the computer (the "banker") roll five dice each. If any die comes up 2 or 5, it **must** be rolled again. You add the pips on dice which did not come up 2 or 5, and then you add to this the total of of the pips from the dice you've rolled again. If a die comes up 2 or 5 the second time, it counts as zero. The aim is to get as close as possible to 9, or to get a two-digit number ending in 9. The program automatically strips a two-digit number down to its final digit. RUN it a few times, and you'll begin to see why this game is so popular. The program allows 9 winning games, with dead-heats (or "stand-offs") not counted. The winner is the player with the most games out of 9.

```
5 PRINT"□"
10 B1=0
20 P1=0
30 GOTO340
40 D=0
50 C=0
60 FORG=1TO5
70 A=INT(6*RND(1))+1
80 IFA=2ORA=5THENC=C+1
90 IFA=2ORA=5THENA=0
```

```
100 PRINTA;" ";
110 D=D+A
120 NEXTG
130 PRINT
140 PRINTD,
150 IFD>9THEND=D-10
160 PRINTD
170 IFD>9THEND=D-10
180 PRINT"TOTAL ON FIRST ROLL IS";D
190 IFD=9THENPRINT"LA GRANDE"
200 IFD=8THENPRINT"LA PETITE"
210 IFD=7THENPRINT"NATURAL"
220 IFC=0ORD=7ORD=8ORD=9THENRETURN
230 PRINT"MUST ROLL ";C;"AGAIN"
240 FORA=1TOC
250 E=INT(6*RND(1))+1
260 IFE=2ORE=5THENE=0
270 D=D+E
280 NEXTA
290 PRINTD,
300 IFD>9THEND=D-10
310 PRINTD
320 IFD>9THEND=D-10
330 RETURN
340 PRINT"BANKER"
350 GOSUB40
360 PRINT"FINAL TOTAL";D
370 INPUTA$
380 J=D
390 PRINT"PLAYER"
400 GOSUB40
410 INPUTA$
420 PRINT"BANKER","PLAYER"
430 PRINTJ,D
440 IFJ=DTHENPRINT,"STAND OFF"
450 IFJ=DTHENGOTO510
460 IFJ>DTHENPRINT"BANKER";
470 IFJ>DTHENB1=B1+1
480 IFJ<DTHENPRINT"PLAYER";
490 IFJ<DTHENP1=P1+1
500 PRINT" WINS"
```

```
510 PRINT"TOTALS"
520 PRINTB1,P1
530 IFB1+P1=9THENEND
540 INPUTA$
550 PRINT"□"
560 GOTO340
```



Craps

In THE COMPLETE BOOK OF DICE GAMES, Skip Frey describes Craps as "the premier dice game". According to Mr. Frey, "it is played everywhere from back alleys to posh casinos in Las Vegas and Monte Carlo". Despite this glowing description, it can become a very dull game indeed when player with a computer. After all, to play the game you just roll dice, and if you have your trusty computer doing this for you, there isn't much else to do. Therefore, we've jazzed up the program a bit, to save you falling asleep at the keyboard. This Craps program gives you a starting stake of $20, and then adds to it, or takes away, in accordance with your luck with the dice.

OFFICIAL TERMS FOR CRAPS:

NATURAL — A 7 or an 11 on the first roll is a NATURAL. Roll this, and you win.

CRAPS. — A 2, 3 or 12 on the first roll is CRAPS. Roll this, and you've lost.

POINT — A 4, 5, 6, 8, 9 or 10 on the first roll becomes your POINT. In the program, the variable E is your point.

If you don't roll a natural or craps, you continue to roll until you 'make your point'. In this game, see line 270, you win a grand sum if you make your point. However, and this is a big however, if you throw a 7 before you make your point, you lose (line 120). The program subtracts $3, plus a dollar for every roll of the dice you've made in that game. So long as you manage to end a game with $1 or more, the computer will offer you a new game.

```
5 M=20:PRINT"□"
10 A=0
20 E=0
30 INPUT"PRESS RETURN TO ROLL";A$
50 GOSUB320
60 B=INT(6*RND(1))+1
70 C=INT(6*RND(1))+1
80 D=B+C
90 A=A+1
95 IFA=6THENPRINT"□"
97 IFA=6THENA=2
100 IFA=1THEN150
110 IFD=ETHEN270
120 IFD=7THEN300
130 PRINT"THE DICE CAME UP";B;C;"TOTAL";D
140 GOTO30
150 IFD=7ORD=11THEN190
160 IFD<4ORD=12THEN210
170 E=D
180 GOTO130
190 PRINT"YOU ROLLED";D;"SO YOU WIN"
195 M=M+5+A
200 GOTO220
210 PRINT"FATE GAVE YOU";D,"SO YOU LOSE"
220 PRINT"YOU'VE GOT $";M
222 IFM<1THENSTOP
225 INPUT"ANOTHER GAME";A$
230 PRINT"□"
250 IFA$<>"NO"THEN10
260 END
270 PRINT"THAT TIME YOU GOT";D
280 M=M+5+A
290 GOTO220
300 PRINT"FOOL,YOU BLEW IT BY   ROLLING";D
```

```
305 M=M-3-A
310 GOTO220
320 FORF=1TO4000
330 NEXTF
340 RETURN
```

Life Expectancy

There was a light-hearted life expectancy program earlier in the book. The next program, based loosely on actuarial tables, is closer to a "serious" life expectancy program.

```
1 C=0
2 PRINT"LIVES"
3 A=71
4 INPUT"AGE(YRS)";B
6 GOSUB54
7 INPUT"MARRIED";A$
9 GOSUB54
10 IFA$="YES"THENA=76
11 PRINT"HAVE YOU BEEN RICH    MOST OF YOUR"
12 INPUT"LIFE(Y OR N)";C$
14 GOSUB54
15 IFC$="Y"THENA=A-3
16 INPUT"ARE YOU OVERWEIGHT";D$
18 GOSUB54
19 IFD$="NO"ORB<40THEN24
20 P=0
21 IFD$="YES"THENINPUT"BY HOW MANY POUNDS";P
22 C=C+P
23 GOSUB54
24 PRINT"EXERCISE.NEVER,","SOMETIMES,OFTEN"
25 INPUTE$
26 GOSUB54
27 IFE$="SOMETIMES"THENA=A+3
28 IFE$="OFTEN"THENA=A+5
29 GOSUB54
30 INPUT"ARE YOU OFTEN TENSE";F$
32 GOSUB54
```

```
33 IFF$="YES"THENA=A-3
34 IFF$="NO"THENA=A+3
35 PRINT"DRINK:LITTLE(0),MOD.(5)"
36 INPUT"HEAVY(10)";G
38 GOSUB54
39 INPUT"DO YOU SMOKE";H$
40 IFH$="YES"THENA=A-5
42 GOSUB54
43 INPUT"OFTEN ILL";K$
45 IFK$="YES"THENA=A-3
46 IFK$="NO"THENA=A+3
47 GOSUB54
48 PRINT"ESTIMATED AGE OF DEATH"
49 PRINT
50 IFB>ATHENA=B+INT(B-A)/2)
51 PRINT,"FEMALE-";A+7-C/5-G
52 PRINT,"MALE-";A-C/5-G
53 END
54 FORJ=1TO5
55 PRINT
56 NEXT:RETURN
```

Bird Cage

In its first incarnation, in England, this game had the most improbable name of Sweat-Cloth, and when exported to the United States in the early years of the 19th Century, it became known first as Sweat. Its name changed through the years to Chucker-Luck, Chuck-Luck, Chuck-a-Luck or just plain Chuck. Nowadays, because of the equipment used in the non-computer version, the game is often called The Bird Cage.

The bird cage is an enclosed wire cage holding three dice. Players bet on the likelihood of a particular number coming up. If, for example, they place their money on six, and one of the three dice ends up with a six showing, they get their money back. If all three dice show six, then they get three times their money. A fairly simple game, but one which arouses passion among bird cage devotees.

This program does most of the work for you. When it first asks SIZE OF BET? you enter the number of pounds and pence you wish to bet. At all times you know how much money you have left (you start with £30) and you can bet up to the total you have. You automatically lose this amount as the game begins (so one die coming up with your number just returns your

money, you need two or three to show a profit). Your next prompt will be NUMBER? and here the computer just wants you to enter a number between one and six. The game explains itself as the program progresses.

```
10 M=30:PRINT"□"
```

M is your starting stake, and M your running cash total

```
20 GOSUB280
```

Subroutine 280 prints your money on the screen, and incorporates a delay loop (290 and 300) which is accessed separately at one point (line 220)

```
30 INPUT"SIZE OF BET";A
50 IFA>MTHEN30
60 PRINT"£";A
70 M=M-A
80 INPUT"NUMBER";B
110 IFB<1ORB>6THEN80
```

The next section rolls the dice, changing your stake as it does so if you win.

```
120 FORC=1TO3
130 W=0
```

W is the win/lose flag and paymaster

```
140 GOSUB290
150 D=INT(6*RND(1))+1
155 PRINT
160 PRINT"DICE NO.";C;"FELL";D
170 IFD=BTHENW=A
180 IFD=BTHENPRINT,"WIN £";W
190 M=M+W
```

The subroutine changes the total if you have won

```
200 GOSUB280
210 NEXTC
```

There is a short delay before the game continues

```
220 GOSUB290
230 PRINT"□"
```

If you have money, action moves back to line 20

```
240 IFM>0THEN20
```

If you are broke, woe...

```
250 PRINT"GAME OVER,YOU ARE      BROKE"
270 GOTO250
```

This is the tally-changer and delay subroutine

```
280 PRINT"STAKE £";M
290 FORN=1TO2000
300 NEXTN
310 RETURN
```

Seventh Heaven

In its youth as a dice game, this was known as UNDER AND OVER 7. It seems very attractive to gamblers, because it appears loaded in the player's favour. As you'll discover when you play it with 'money' which only exists in the variable store, the game is loaded against the player. If the random number generator worked perfectly, and you played this game for ever, your losses would outweigh your gains by 15⅔%. Now you've been warned, we'll move into SEVENTH HEAVEN.

Once again, we've used M to represent your money.

```
10 PRINT"□"
20 M=30
```

And once again, we have a subroutine to print out your growing (7) total

```
30 GOSUB320
```

The next section asks you to place your bet on one of the three possibilities: the dice will land with a total under 7 (A); equal to 7 (B); or over 7 (C). You indicate your choice by entering either A, B or C.

```
40 PRINT"PLACE YOUR BET"
50 PRINT"(A) UNDER 7 (B) 7","(C) OVER 7"
```

The next line explains your winnings — money back (A or C) or 4 to 1 (B, written as 5 for 1 to look as if you are getting more...sneaky)

```
60 PRINT"(A) EVEN (B) 5 TO 1","(C) EVEN"
70 INPUTA$
```

If you wish to stop the game, enter "S"

```
80 IFA$="S"THEN END
```

Now, you enter the amount of your bet

```
90 INPUT"AMOUNT";A
110 IFA>MTHEN90
130 B=INT(6*RND(1))+1
150 C=INT(6*RND(1))+1
170 D=C+B
180 PRINTD
```

This section of the program works out how well, or otherwise, you have done

```
185 W=-A
190 IFD=7ANDA$="B"THENW=4*A
200 IFD<7ANDA$="A"THENW=A
210 IFD>7ANDA$="C"THENW=A
230 M=M+W
240 IFW>0THENPRINT"YOU WIN $";W
250 IFW<0THENPRINT"YOU LOSE $";-W
260 GOSUB320
270 FORN=1TO1000
280 NEXTN
290 IFM<1THENEND
300 PRINT"□"
310 GOTO30
320 PRINT"STAKE $";M:RETURN
```

Let the longer games begin

After a while, you'll want to write longer games. When you do this, you'll discover that many of the good habits you've learned can desert you. It is

very easy to set up a long and sloppy set of IF/THENs which could easily be replaced by an IF/THEN instruction to GOSUB. When you have memory to spare, it often seems too much trouble to bother cleaning up your programs. Unused subroutines clutter up the bottom ends of your programs. GOTO statements cover a multitude of situations which arose because you did not give sufficient thought to the maximum line number you would need.

If you are going to take up flow-charting, now is the time to begin. If you can't be bothered with pretty triangles and things, at least discipline yourself to setting out — on paper — what your program is supposed to do, with arrows linking FOR/NEXT loops, and lines leading to the first lines of subroutines. If you can be bothered, it is worth writing out a full listing for a program once you get it working. Examine it in detail, and you're sure to find more elegant ways of achieving the same ends. Be particularly critical of each and every GOTO command which is non-conditional.

All the programs given so far in this book can act as starter ideas for much bigger and better programs when you get extra money.

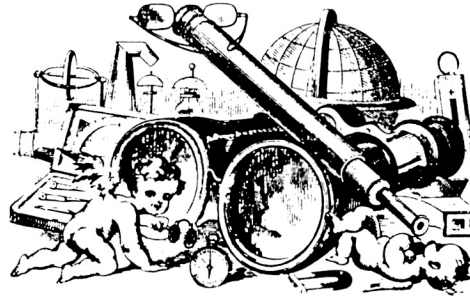
The best thing you can add to a program with added memory is the element of surprise. If you can include situations which do not occur every time a game is played, you'll ensure the game will remain interesting for a much longer time than would be the case if every situation is triggered every time a game is run.

As you know, many of the games in this book run far too fast to be good games without the use of a "delay subroutine". However, as you get into longer games (and I mean ones much longer than those listed in this section) you'll find that slow-running programs and response-times can be boring, especially if you have a graphical element in your program, which "moves" in some way from go to go. Variables which must be assigned at the start of a game can actually be listed right at the end of the program, ending with a GOTO leading back to the start of the game program proper. You can have a line like "DO YOU WANT INSTRUCTIONS?" near the start of a program, and if the answer is "YES" the computer can GOTO the end of the program where the instructions are. LABYRINTH, if you wanted it to run more quickly, could have the instructions at the very end. Doing this would, of course, somewhat defeat the purpose of having a delay subroutine. However, when you write very, very long programs, you will not always want to wait while the computer searches a vast listing for the subroutine.

Another way of improving programs, and making them interesting to players for a longer time, is to use a feature you've seen in some programs, the "degree of difficulty". Make sure that this feature really does increase the difficulty of the game. Ensure that, even at the highest level of play, the final score (or successful landing, or obliterated aliens or whatever) is attainable.

You can add interest to games by awarding points, or scores, or ratings or whatever, that are genuinely related to the speed, skill or whatever the player demonstrated. A further twist is to award a "rank" (like "star fleet captain", "novice" or "incompetent fool") to the player, depending on how well he or she did. Points and ranks ensure that a player remains interested in a game for a longer time, as the player will try to beat his or her previous best score or ranking.

Keeping these features in mind, have a look at the next two games, enter them and run them, and then try to improve on them.



Lunar Landing

```
10 M=0
20 T=0
30 S=0
40 H=5000
50 PRINT"    LUNAR LANDING"
60 F=5000/INT(3*RND(1)+1)
70 Q=-17
80 B=1
100 PRINT
110 PRINT
120 GOTO300
130 PRINT"+ IS TOWARDS LUNA"
140 INPUTZ
150 IFZ<-50ORZ>50THEN410
160 PRINT"FOR HOW MANY":INPUT"SECONDS";E
180 PRINT"□"
190 T=T+E
```

```
0 S=S+10+3*E*((Z+1)/B)
0 F=F-3*E*ABS(Z*INT(3*RND(1))+1)
0 IFF<500THENPRINT"FUEL LOW"
0 H=H-E*S
0 IFH<20ANDH>-10ANDS<12THEN470
0 IFH<=-10THEN440
0 IFF<0THEN440
0 X=INT(10*RND(1))+1
0 IFX=5ANDM<>2THENGOSUB790
0 PRINT
0 PRINTCHR$(96+M);"HEIGHT ABOVE SURFACE:";H
0 IFQ<>-17THENQ=Q-INT(16*RND(1))-1
0 IFQ<0ANDQ>-17THEN440
0 IFQ<>-17THENPRINT"OXYGEN LEFT:";Q
0 PRINTCHR$(96+M);"VELOCITY:";S
0 IFB<>1THENPRINTCHR$(96+M)"WARNING-",,"THRUST ERRATIC"
0 PRINTCHR$(96+M);"FUEL LEFT:";F
0 PRINTCHR$(96+M);"FLIGHT TIME:";T
0 GOSUB510
0 PRINT
0 PRINT"THRUST(-50 TO 50)?"
0 GOTO130
0 PRINT"■CRASH.HIT SURFACE","AT";ABS(S);
0 GOTO440
0 PRINT"■SUCCESSFUL LANDING"
0 PRINT
0 PRINT"FINAL VELOCITY:";ABS(S);
0 GOTO490
0 FORA=1TO22
0 PRINTCHR$(96+M);
0 NEXT
0 RETURN
0 PRINT"■"
0 M=M+1
0 FORV=1TO4
0 PRINT
0 NEXTV
0 U=INT(32000*RND(1))+1
0 FORV=1TO4
0 PRINT"HOUSTON,WE HAVE A     PROBLEM..."
0 PRINTCHR$(INT(32*RND(1)+96));"■DANGER■";CHR$(INT(32*RND(1)+96))
```

```
890 PRINT
900 PRINT
910 PRINT"MALFUNCTION","USE COMPUTER"
920 PRINT"ACCESS CODE";U;"FOR DETAILS"
930 INPUTV
940 PRINT"□"
950 IFU<>VTHEN440
960 ONINT(2*RND(1))+1GOSUB1030,1070
990 INPUT"PRESS RETURN TO RETURNTO FLIGHT";V$
1010 PRINT"□"
1020 RETURN
1030 Q=INT(19*RND(1))+101
1040 PRINT"OXYGEN METER","UNRELIABLE"
1060 RETURN
1070 B=B+INT(3*RND(1))+1
1080 PRINT"THRUST CONTROL ERRATIC"
1100 RETURN
```

Labyrinth

```
10 GOSUB1010
20 PRINT"LABYRINTH"
30 PRINT
40 X=0
50 S=30
60 W=1
70 PRINT"YOU ARE AT THE START"
80 PRINT"OF A LABYRINTH OF MANY";
90 PRINT"TWISTING,TURNING"
100 PRINT"TUNNELS.YOU HAVE 30"
110 PRINT"PIECES OF SILVER.YOU"
120 PRINT"MUST GET TO THE END OF";
130 PRINT"THE LABYRINTH WITH AT"
140 PRINT"LEAST 20 TO PAY THE"
142 PRINT"MINOTAUR"
144 PRINT
150 INPUT"PRESS RETURN";A$
```

```
180 IFA$<>""THENEND
190 GOSUB1010
200 IFW<1THENW=1
210 PRINT"THIS IS MAZE/TUNNEL"
220 PRINT
230 IFW=>10THEN1070
240 PRINT"NUMBER";W;"OF THE","LABYRINTH"
250 PRINT
260 PRINT"(10 IS THE END)"
270 PRINT
280 X=X+1
290 PRINT"THIS IS CHALLENGE","NUMBER";X
300 IFS<1THENS=3
310 PRINT
320 PRINT"YOU HAVE";S,"SILVER PIECES"
330 GOSUB1000
340 PRINT
345 K=INT(4*RND(1))+2
350 PRINT"FACING YOU NOW ARE",K;"DOORS"
360 PRINT"WHICH ONE WILL YOU":INPUT"TRY";A
380 GOSUB1010
390 IFRND(1)<.1THEN690
400 IFA<>KTHEN420
410 IFA=KTHEN690
420 K=INT(4*RND(1))
430 IFK=0THENE$="■RODENTING RAT■"
440 IFK=1THENE$="■WART-FACED WOGGLE■"
450 IFK=2THENE$="■ELLIPSOID OCTOPUS■"
460 IFK=3THENE$="■WACKED-OUT WIZARD■"
470 PRINT"FOOL,YOU'VE WALKED IN ON A"
480 E=INT(4*RND(1))
490 IFE=0THENF$="FLAMING BRAND"
500 IFE=1THENF$="SHINING SWORD"
510 IFE=2THENF$="POISONED NEEDLE"
520 IFE=3THENF$="GOSUB-MACHINE GUN"
530 PRINTE$," ARMED WITH"
540 PRINT"A ";F$
550 PRINT
560 PRINT"WHICH WEAPON DO YOU","CHOOSE?"
570 PRINT
580 PRINT"FLOATING POINT ROM(1),"
```

```
590 PRINT
600 PRINT"FOR/NEXT LOOP(2),"
610 PRINT
620 PRINT"POKED ADDRESS(3)"
630 INPUTB
640 C=INT(3*RND(1))+1
650 GOSUB1010
660 IFB=CTHENGOSUB1170
670 IFB<>CTHENGOSUB1240
680 GOTO150
690 K=INT(4*RND(1))+1
700 ONKGOSUB760,810,850,900
740 GOTO150
750 PRINT
760 PRINT"YOU'VE FALLEN THROUGH"
770 PRINT"    A TRAPDOOR..."
780 W=W-1
790 S=S-INT(2*RND(1))-1
800 RETURN
810 PRINT"A WALL OF FLAME","ENGULFS YOU"
820 W=W-1
830 S=S-INT(2*RND(1))-1
840 RETURN
850 PRINT"THE LOVELY SEMOLINA"
860 PRINT"SOOTHES YOUR FEVERED  BROW"
870 S=S+INT(5*RND(1))+1
880 W=W+INT(3*RND(1))+1
890 RETURN
900 PRINT"JOY OH JOY,A HOARD OF"
910 PRINT"SILVER.CHOOSE UP TO 5 PIECES"
920 PRINT"BUT BE WARNED.THE MORE";
930 PRINT"YOU TAKE,THE MORE IT  WILL"
940 PRINT"COST YOU.HOW MANY?"
950 INPUTD:IFD>5THEN950
960 S=S+D
970 W=W-INT(D/2)
980 RETURN
1000 FORO=1TOINT(1000*RND(1))+8000
1005 NEXT
1010 PRINT"□"
1020 FORI=1TO5
```

```
1030 PRINT
1040 NEXT
1050 RETURN
1060 IFW<>10THENRETURN
1070 PRINT"YOU ARE AT THE END"
1080 PRINT"DO YOU HAVE ENOUGH","SILVER?"
1090 PRINT"PRESS RETURN TO","FIND OUT"
1100 INPUTC$
1110 IFS<20THENPRINT"THE MINOTAUR HAS EATENYOU"
1120 IFS<20THEN1110
1130 PRINT"YES,YOU HAVE";S,"SILVER PIECES"
1140 PRINT"YOU HAVE WON"
1150 GOTO1130
1170 PRINT"YOU BEAT THE",E$
1180 S=S+INT(3*RND(1))+1
1190 PRINT"AND HAVE";S,"SILVER PIECES"
1200 W=W+INT(3*RND(1))+1
1210 PRINT
1220 PRINT"YOU ARE APPROACHING","SECTOR";W
1230 RETURN
1240 PRINT"THE ";E$,"BEAT YOU,AND"
1250 S=S-INT(4*RND(1))-1
1260 PRINT"LEFT YOU WITH";S,"SILVER PIECES"
1270 W=W-1
1280 IFW<1THENW=1
1290 PRINT"AND SENT YOU BACK TO",W
1300 RETURN
```

Teacher

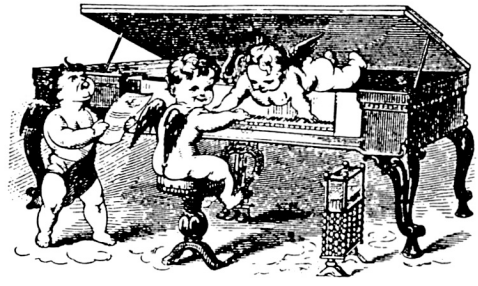
The computer is an effective game player. Writing and running programs on the computer enhances programming skills. However, it can also be used in a direct role as a teaching aid. Its main use is in the field of quizzes. While the computer can only select from a list of non-numerical questions, the computer can easliy be reprogrammed to create its own numerical questions.

Another use in teaching is for the computer to create lists and tables. We will look at this use first.

```
1 PRINT
2 PRINT
3 PRINT"MULTIPLICATION TABLES"
```

```
4 PRINT
5 PRINT
6 PRINT
7 PRINT"WHICH TIMES TABLE","WOULD YOU"
8 INPUT"LIKE ME TO PRINT";A
10 PRINT"□"
11 FORJ=1TO12
12 PRINTJ;"X";A;"=";A*J
13 NEXTJ
14 PRINT
15 PRINT
16 PRINT"DO YOU WANT ANOTHER":INPUT"GO";A$
18 PRINT"□"
19 IFA$<>"NO"THEN4
20 FORS=1TO5
21 PRINT
22 NEXTS
23 PRINT"OK,BYE FOR NOW"
24 END
```

A far more useful table is produced by the following program (and the display is a little more imaginitive). Again, this program is given to suggest ideas for your own programs.



A Degree of Conversion

```
20 PRINT"□A DEGREE OF CONVERSION"
30 K=INT(32*RND(1)+96)
40 FORS=1TO22
50 PRINTCHR$(K);
60 NEXTS
70 PRINT
80 PRINT
90 PRINT
100 PRINT
110 PRINT"WHAT IS THE LOWEST    TEMP.(F) YOU"
120 INPUT"WANT TO CONVERT";A
140 INPUT"AND HIGHEST";B
160 IFB<ATHENE=A
170 IFA<BTHENF=A
180 IFB<ATHENF=B
190 IFA<BTHENE=B
200 INPUT"IN WHAT DEGREE STEPS";Z
230 PRINT"□ F   C   K"
240 PRINT
250 H=INT(32*RND(1)+96)
260 FORU=1TO7
270 PRINTCHR$(H);
280 NEXTU
290 PRINT
300 FORD=FTOE+ZSTEPZ
310 C=INT(5*(D-32)/9+.5)
320 K=C+273
330 PRINTD;C;K
340 NEXTD
350 END
```

In all programs, and especially in educational ones, you have to try and anticipate any mistakes (deliberate or otherwise) users will make when running a program. The lines 160 to 190 cover the possibility that a user will input the higher temperature first, rather than the lower one which was requested.

There is no reason why educational programs should not have displays that are as attractive as games programs, so lines 30 to 60 and lines 250 to 280 select, in effect, a random underline. It is a good idea to incorporate such features whenever you have sufficient memory.

Line 300 tells the VIC to print the value above the maximum requested, in order to ensure that the required temperature is covered.

This is quite an interesting program to run, especially if you select an enormous range of temperatures, and select an unexpected increment (like 117 degrees). Of course, the computer deals as easily with these cases as it does with the more predictable 0 degrees to 100 degrees, but it seems a little surprising to first-time users that it does so.

As an exercise, modify the program so it will not print any value but 0 if the temperature drops below absolute zero (0 Kelvin, or minus 273C).

We will look now at a numerical quiz, in which the VIC creates the questions, checks their correctness, and then gives the user a score.

Multiplication Quiz

```
1 H=0
2 GOSUB46
3 PRINT"■MULTIPLICATION QUIZ"
4 GOSUB46
5 PRINT"DEGREE OF DIFICULTY":INPUT"(1 TO 10)";A
7 IFA<1ORA>10THEN1
8 A=INT(A/2+.5)
10 GOSUB46
11 INPUT"HOW MANY QUESTIONS";B
13 IFB<1THEN11
14 PRINT"■"
15 FORG=1TOB
16 C=A*INT(10*RND(1)+1)
17 D=A*INT(10*RND(1)+1)
18 E=C*D
19 GOSUB46
20 PRINT"QUESTION NUMBER";G
21 GOSUB46
22 PRINT"WHAT IS";C;"TIMES";D
23 INPUTF
```

```
25 GOSUB46
26 IFF=ETHEN42
27 PRINT"INCORRECT.":PRINT"THE ANSWER IS";E
28 GOSUB46
29 PRINT"YOUR SCORE IS";H
30 PRINT"OUT OF";G
31 PRINT"PRESS RETURN"
32 IFG<>BTHENPRINT" TO CONTINUEE"
33 INPUTA$
34 PRINT"□"
35 NEXTG:G=G-1
36 FORK=1TO2
37 GOSUB46
38 NEXTK
39 PRINT"END OF QUIZ.YOUR SCOREIS"
40 PRINTINT(H*100/G);"%"
41 END
42 H=H+1
43 GOSUB46
44 PRINT"CORRECT.THE ANSWER IS";E
45 GOTO28
46 FORS=1TO5
47 PRINT
48 NEXTS
49 RETURN
```

To make this an addition, subtract or division quiz, simply change the arithmetic operation specified in line 18, and the word (TIMES in the above program) in line 22.

There are a number of things which can be learned from this program. The most obvious is the use of the subroutine (GOSUB 46) which spaces the question and answer across the screen, enhancing readability. Lines 7 and 13 ask the questions in lines 5 and 11 again if an answer in the required range is not given the first time. There is no reason why you cannot put in an additional line which says something like ONLY ANSWERS BETWEEN 1 AND 10 ARE ACCEPTABLE if you want to. Line 8 can be omitted if you want very difficult quizzes to be available. The one to 10 option in line 5 is to give the user the impression that he or she has greater influence on the program than in fact is the case.

At the end of all questions in a set, but the last, lines 31 and 32 cause PRESS RETURN TO CONTINUE to appear at the bottom of the display.

Line 32 ensures that after the final question only the words PRESS RETURN appear.

Squares

The following table prints out numbers and their squares.

```
1 DEF FNS(Z)=Z*Z
2 GOSUB22
3 PRINT,"SQUARES"
4 GOSUB22
5 INPUT"LOWEST NUMBER";A
7 GOSUB22
8 INPUT"HIGHEST NUMBER";B
10 GOSUB22
11 INPUT"PRESS RETURN FOR TABLE";U$
13 PRINT"□"
14 FORX=ATOB
17 PRINTX;"SQUARED IS";FNS(X)
18 NEXTX
19 PRINT
20 PRINT"      END OF TABLE"
21 END
22 FORC=1TO3
23 PRINT
24 NEXTC
25 RETURN
```

Notice how the "squared" function is DEFINED in line one, and then USED in line 17. You could put different functions in line one if you wanted, (although you'd have to change the word "SQUARED" in line 17.)

I'm sure you will now be able to work out an endless stream of numerical quizzes for yourself, your children or your class to tackle. Non-numerical quizzes are very useful, but they require much more work in programming. Whereas the computer can create its own numerical questions, each non-numerical question must be specified, and each answer included in full in the program.

Appendices

APPENDIX 1 — COMMANDS

CONT	re-start the program after it has been stopped by END or STOP.
LIST	list the current program.
LOAD	get program from tape (or disc).
NEW	erase current program.
RUN	begin execution of a program.
SAVE	store program on tape (or disc).
VERIFY	check that program stored on tape (or disc) is the same as the current program.

APPENDIX 2 — STATEMENTS

CLOSE	complete and close an opened file.
CLR	delete all variables.
CMD	reroute the PRINT output to some other device.
DATA	a list of items to be used by READ statements.
DEF	define a user-defined-function. Function name must begin with FN.
DIM	create an array of specified size.
END	end of program.
FOR...TO	begin a loop. The specified variable is incremented each time round the loop. If no step size is given the step is automatically one. The loop is repeated until the variable falls outside the given range of values.
GET	scan the keyboard and assign the specified variable with the contents of the key currently being pressed.
GOSUB	as GOTO except that the next line to be executed is "remembered" and used by later RETURN statement.
GOTO	control moves to the specified line number.
IF...THEN	IF the condition given is true THEN carry out the required instruction, otherwise move to the next line.
INPUT	wait for a value to be typed in from the keyboard and assign it to a variable.
ON	used with GOTO or GOSUB to choose from one or more possible destinations depending on the value of an expression.
OPEN	open a file or device.
POKE	change the contents of a given address to a given number.
PRINT	display information on the screen.
READ	assign to a variable the value found in a corresponding DATA statement.
REM	this line is ignored by the VIC.
RESTORE	move the DATA pointer to the first item in the list.
RETURN	control moves to the line immediately after the last GOSUB statement executed.

STOP	stops the program with the message BREAK ERROR IN LINE.
SYS	call machine code subroutine beginning at specified address.
WAIT	pause until a specified memory location changes to a specified value or binary bit pattern.

APPENDIX 3 — FUNCTIONS

ABS(X)	X if X is greater than or equal to zero, or −X if X is less than zero. The answer is always positive.
ATN(X)	The arctangent of X, in radians.
COS(X)	The cosine of X radians.
EXP(X)	The value e raised to the power of X. e is approximately 2.71827183
FN__(X)	User-defined-function determined by a previous DEF statement.
INT(X)	The largest integer which is not greater than X.
LOG(X)	The natural logarithm of X.
PEEK(X)	The contents of address X.
RND(X)	A random number between zero and one.
SIN(X)	The sine of X radians.
SQR(X)	The number, which when multiplied by itself equals X.
TAN(X)	The tangent of X radians.
USR(X)	Call the machine code subroutine beginning at address stored in locations 1 and 2. The value of X is passed to the machine code program.
ASC(A$)	The character code of the first character of A$;
CHR$(X)	The character whose code is X X.
LEFT$(A$,X)	The first X characters of A$.
LEN(A$)	The number of characters in A$.
MID$(A$,X,Y)	The first Y characters of A$ starting with the Xth character.
RIGHT$(A$,X)	The last X characters of A$.
STR$(X)	The number X in the form of a string.
VAL(A$)	The number contained in the string A$
FRE(X)	The number of available unused bytes.
POS(X)	The column number of the next print position.
SPC(X)	A string of X spaces. (May only be used in PRINT).
TAB(X)	Moves the print position to column X. (May only be used in PRINT).

Appendix 4 — Screen Codes

SET 1	SET 2	POKE	SET 1	SET 2	POKE	SET 1	SET 2	POKE
@		0	U	u	21	*		42
A	a	1	V	v	22	+		43
B	b	2	W	w	23	,		44
C	c	3	X	x	24	—		45
D	d	4	Y	y	25	.		46
E	e	5	Z	z	26	/		47
F	f	6	[		27	ø		48
G	g	7	£		28	1		49
H	h	8	]		29	2		50
I	i	9	↑		30	3		51
J	j	10	←		31	4		52
K	k	11	SPACE		32	5		53
L	l	12	!		33	6		54
M	m	13	"		34	7		55
N	n	14	#		35	8		56
O	o	15	$		36	9		57
P	p	16	%		37	:		58
Q	q	17	&		38	;		59
R	r	18	'		39	<		60
S	s	19	(		40	=		61
T	t	20	)		41	>		62

SET 1	SET 2	POKE	SET 1	SET 2	POKE	SET 1	SET 2	POKE
?		63		T	84			106
		64		U	85			107
♠	A	65		V	86			108
	B	66		W	87			109
	C	67	♣	X	88			110
	D	68		Y	89			111
	E	69	♦	Z	90			112
	F	70			91			113
	G	71			92			114
	H	72			93			115
	I	73			94			116
	J	74			95			117
	K	75	SPACE		96			118
	L	76			97			119
	M	77			98			120
	N	78			99			121
	O	79			100		✓	122
	P	80			101			123
●	Q	81			102			124
	R	82			103			125
♥	S	83			104			126
					105			127

Appendix 5 — ASCII and CHR$ Codes

PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$
	00		10	SPACE	20	Ø	30
	01	CRSR ↓	11	!	21	1	31
	02	RVS ON	12	"	22	2	32
	03	CLR HOME	13	#	23	3	33
	04	INST DEL	14	$	24	4	34
WHT	05		15	%	25	5	35
	06		16	&	26	6	36
	07		17	'	27	7	37
	08		18	(	28	8	38
	09		19	)	29	9	39
	0A		1A	*	2A	:	3A
	0B		1B	+	2B	;	3B
	0C	RED	1C	,	2C	<	3C
RETURN	0D	CRSR →	1D	-	2D	=	3D
SWITCH TO LOWER CASE	0E	GRN	1E	.	2E	>	3E
	0F	BLU	1F	/	2F	?	3F

PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$
@	64	U	85		106		127
A	65	V	86		107		128
B	66	W	87		108		129
C	67	X	88		109		130
D	68	Y	89		110		131
E	69	Z	90		111		132
F	70	[	91		112	f1	133
G	71	£	92		113	f3	134
H	72	]	93		114	f5	135
I	73	↑	94		115	f7	136
J	74	←	95		111	f2	137
K	75		96		117	f4	138
I	76		97		118	f6	139
M	77		98		119	f8	140
N	78		99		120	SHIFT RETURN	141
O	79		100		121	SWITCH TO UPPER CASE	142
P	80		101		122		143
Q	81		102		123	BLK	144
R	82		103		124	CRSR	145
S	83		104		125	RVS OFF	146
T	84		105		126	CLR HOME	147

PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$
INST DEL	148	CYN	159		170		181
	149	SPACE	160		171		182
	150		161		172		183
	151		162		173		184
	152		163		174		185
	153		164		175		186
	154		165		176		187
	155		166		177		188
PUR	156		167		178		189
CRSR	157		168		179		190
YEL	158		169		180		191

Index

Photo-typeset and printed in England by Commercial Colour Press, London E.7.

And now that you've got your VIC, what do you do with it?

This book contains the answer. In fact, this book contains over 60 answers. Over 60 great games programs, including:

- LIFE
- BLACKJACK
- CUBIK
- FRUIT MACHINE
- LOST IN SPACE
- LABYRINTH
- LUNAR LANDING

But it is not all games. As well as the programs, the book leads you through most of the VIC's commands and statements, step by simple step.

Whether you've never touched a computer before you bought your VIC, or you've had a lot of experience, you'll find much of interest and value in this book by Tim Hartnell.

Tim's previous computer books have been warmly welcomed by the computer press.

> PERSONAL COMPUTER WORLD commented: "If, in any sense you are a beginner to programming of computing, this is *undoubtably* the book to read. Full of insight, witty, sensible and extremely funny, it eases you into programming practically from the word go ... a racy but informative style, obviously aimed at kids of all ages. The problem for me was to borrow the book long enough from my son (aged eight) to review it ..."
>
> And COMPUTING TODAY said "... Tim Hartnell has certainly provided the reader with many varied programs but in the text, linked to most of the listings, is a well thought out "hands on" learning approach ... As you work your way through the book, not only does your library of programs grow but also your understanding of the BASIC commands which make them possible ..."

If you've got a VIC, you need this book. Could you live another hour without listening to your computer composing a symphony, without fighting your way out of the Labyrinth, or surviving a brain-boiling round of FRENZY?

THIS BOOK IS THE KEY TO MAKING THE MOST OF YOUR VIC 20.

Another great book from
INTERFACE